

IN THIS ISSUE: THREADS SAMPLES AND DOCUMENTATION!

MacTech Formerly MacTutor MAGAZINE™

FOR MACINTOSH PROGRAMMERS & DEVELOPERS

NOVEMBER 1994 • VOLUME 10, No. 11

In This Issue!

GETTING STARTED:
Working With Color, Part II

SIDE INFORMATION:
Link Like A Moviemaker, Part II

VISUAL PROGRAMMING
Graph CPX - Tutorial

PROGRAMMER CHALLENGE:
Huffman Decoding
Cribble Revisited

IMPLEMENTOR'S JOURNAL
Implementing Elegant Drag & Drop for Styled Text Fields

ESSENTIAL APPLE TECHNOLOGY
Threading Your Apps

THINK TOP 10

ESSENTIAL APPLE TECHNOLOGY
Full Thread Manager Documentation

AND MORE!



Even Better Bus Error

\$5.85 US
\$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

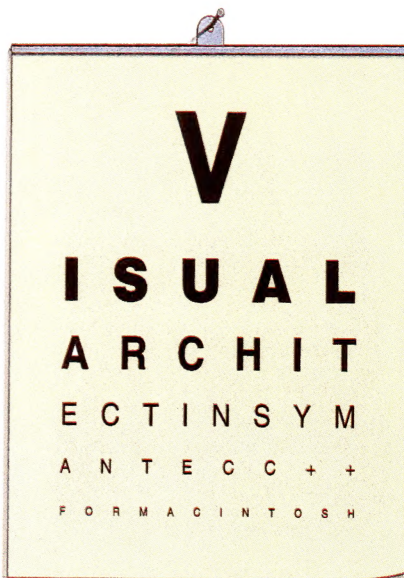
We've got something to show you that you will have to see with your own eyes to believe.

It's Symantec C++ 7.0. And if you're a Macintosh developer, you'll want to take a good look.

VISUAL ARCHITECT. THE EASIEST WAY TO DESIGN A MACINTOSH INTERFACE.

Our new Visual Architect lets you visually design and create all of your windows, dialogs, bitmaps, alerts, controls and menus. Then it automatically generates your complete Mac application.

You can test the user interaction



Any way you look at it, Symantec C++ 7.0 for Macintosh with its Visual Architect is the easiest way to design the Macintosh user interface.

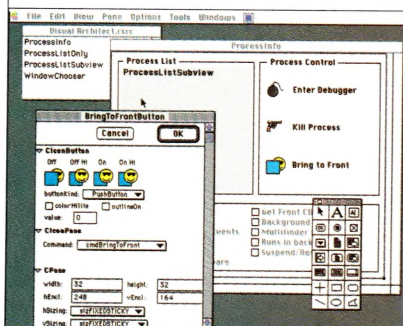
code specific to 68K and it also supports Power Macintosh procedural pointers.

So all of your TCL applications written in Symantec C++ 7.0 will port easily to Power Macintosh giving you a head start on your Power Macintosh development.

What's more, we're offering you a developers' pre-production release of our Power Macintosh Cross Development Kit. It has the tools you need to get started on creating native Power Macintosh applications today, including a copy of Apple's Power Macintosh linker and debugger.

NEW SYMANTEC C++ 7.0. YOU'D HAVE TO BE BLIND TO PROGRAM MACINTOSH WITH ANYTHING ELSE.

of your Macintosh application immediately while in Visual



Visual Architect simplifies the creation of your Macintosh interface so you can focus on the big picture: creating killer applications.

Architect's prototype mode, without even having to compile your application.

Completely integrated within our THINK Project Manager environment, Visual Architect is immediately accessible to you every step of the way. Simplifying

your entire development process from design to the final code.

To help cut your development cycle even further, we've also included a brand new object browser that more efficiently lists, examines and modifies all objects instantiated during the course of your applications development.

We've also enhanced our THINK Class Library 2.0 (TCL 2.0) with C++ pointers and memory management, new support for AppleEvents and scriptable apps, exception handling support and of course, object-persistence.

THE PATH TO POWER MACINTOSH.

Our new TCL 2.0 uses universal headers, removes all the assembly

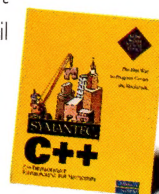
It's a way to speed up the development of your Macintosh applications right now – and be the first to move your programs to the Power Macintosh platform.

CALL 1-800-628-4777.

Ask for ext.9H17 to upgrade to Symantec C++ 7.0 for Macintosh today for just \$149.95* and get our developers' pre-production release of the Power Macintosh Cross Development Kit for just \$100.

This offer is not available in any retail store. So call now.

And get your code ready for Power Macintosh.



SYMANTEC®

*\$499.00 Suggested Retail Price. Power Macintosh Development Kit available only with purchase of or upgrade to Symantec C++ 7.0. For more information in Canada, call 1-800-667-8661. Symantec, the Symantec logo, Symantec C++, and Visual Architect are trademarks of Symantec Corporation. All other products or brand names are trademarks of their respective holders. ©1994, Symantec Corporation. All rights reserved.

Eddy Award Winner for Best New Developer Tool
– MacUser Editors Choice Awards, 1993

“A distinct improvement over ResEdit.”
– MacTech / MacTutor

“Resorcerer’s data template system is amazing!”
– Bill Goodman, author of Compact Pro

“Nuke ResEdit! Resorcerer is mission-critical for us.”
– Dave Winer, Userland Frontier

“The color pixel editors are wonderful! A work of art!”
– Dave Winzler, author of Microseeds Redux

“Every Macintosh developer should own a copy of Resorcerer.”
– Leonard Rosenthol, Aladdin Systems

“Resorcerer will pay for itself many times over in saved time and effort.”
– MacUser review

“The template that disassembles PICT’s is awesome!”
– Bill Steinberg, author of Pyro! and PBTools

“Resorcerer proved indispensable in its own creation!”
– Doug McKenna, author of Resorcerer

“...a wealth of time-saving tools.”
MacUser Review, Dec. 1992



RESORCERER[®]

Version 1.2.4

ORDERING INFO

Needs: ≥Mac Plus, ≥ Sys 4.2, 1MB
Likes: ≥Mac Plus, ≥ Sys 7.0, 2MB
32-bit clean, AU/X compatible

Price: \$256 (decimal)
(Educational, quantity, or
other discounts available)

Includes: 500 page manual
60-day Money-Back Guarantee
Domestic UPS ground shipping

Payment: Check, PO's, or Visa/MC

Extras (call us):
COD, FedEx, UPS Blue/Red,
International Shipping

Downloadable Demos/Updaters:

AppleLink: Software Sampler
AOL: Software Libs/Development
CompuServe: MACDEV/Tools
or call us.

The Resource Editor for the Macintosh Wizard

- New 1.2 Features:**
- New 'cicn', 'ppat', 'crsr', 'acur', 'pltt', 'clut' editors
 - Powerful icon family editing (all 9 icon types)
 - Color pixel anti-aliasing, dithering, and lots more
 - Complete PICT disassembly and reassembly
 - Resource sorting; ROM resource browsing
 - 120 template field parsing types now supported
 - New insertion & deletion template field types
 - Text-only PICT resources
 - Lots of improvements throughout

- Easier, faster, more Mac-like, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Help, AppleEvent, OCE, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Well-designed, helpful developer tools being added all the time
- Relied on by thousands of Macintosh developers around the world

MATHEMÆSTHETICS, INC.

P.O. Box 298 • Boulder • CO • 80306-0298 • USA

Phone: (303) 440-0707 • Fax: (303) 440-0504

AppleLink/AmericaOnline: RESORCERER • Internet: resorcerer@aol.com

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**? If you have any questions, feel free to call us at 310/575-4343 or fax us at 310/575-0925.

DEPARTMENTS	Internet	CompuServe	AppleLink	America Online	Genie
Orders, Circulation, & Customer Service	custservice@xplain.com	71333,1063	MT.CUSTSVC	MT CUSTSVC	MACTECHMAG
Editorial	editorial@xplain.com	71333,1065	MT.EDITORIAL	MT EDITORS	—
Programmer's Challenge	progchallenge@xplain.com	71552,174	MT.PROGCHAL	MT PRGCHAL	—
Ad Sales	adsales@xplain.com	71552,172	MT.ADSALES	MT ADSALES	—
Accounting	accounting@xplain.com	—	—	—	—
Marketing	marketing@xplain.com	—	—	—	—
Press Releases	pressreleases@xplain.com	—	—	—	—
General	info@xplain.com	71333,1064	MACTECHMAG	MacTechMag	—
Online support area	ftp://ftp.netcom.com/pub/xplain	type GO MACTECHMAG	see Third Parties: Third Parties (H-O)	use keyword: MACTECHMAG	—

MacTECH MAGAZINE

Publisher Emeritus • David Williams
Editor-in-Chief/Publisher • Neil Ticktin
Editor • Scott T Boyd
Assistant Editor • Mary Elaine Califf
Technical Editor • Ken Gladstone
Advertising Executive • Ruth Subrin
Art Direction • Judith Chaplin, Chaplin & Assoc.

XPLAIN CORPORATION

VP Finance & Operations • Andrea J. Sniderman
Customer Service • Al Estrada
Software Engineer • Don Bresee
Accounting Assistant • Brian Shin
Administrative Assistant • Susan Pomrantz
Board of Advisors • Blake Park, Alan Carsrud

AUTHORS & REGULAR CONTRIBUTORS



MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology. We are dedicated to the distribution of useful programming information without regard to Apple's developer status. For information on submitting articles, ask us for our **writer's kit** which includes the terms and conditions upon which we publish articles.

Richard Clark
 General Magic
 Mountain View, CA

Chris Espinosa
 Apple Computer, Inc.
 Cupertino, CA

Jörg Langowski
 Jörg's Folder
 Grenoble Cedex, France

David R. Mark
 M/MAC
 Arlington, VA

Jordan Mattson
 Apple Computer, Inc.
 Cupertino, CA

Mike Scanlin
 Programmer's Challenge
 Mountain View, CA

**Symantec Technical
 Support Group**
 THINK Division
 Eugene, OR

The names MacTech, MacTech Magazine, MacTutor and the MacTutorMan logo are registered trademarks of Xplain Corporation. All contents are copyright 1991-1994 by Xplain Corporation. All rights reserved. Trademarks appearing in MacTech Magazine remain the property of the companies that hold license.

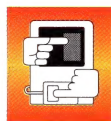


Printed on recycled paper.



MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 1617 Pontius Avenue, 2nd Floor, Los Angeles, CA 90025-9555. Voice: 310/575-4343, FAX: 310/575-0925. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Second Class postage is paid at Los Angeles, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 250055, Los Angeles, CA 90025-9555.



GETTING STARTED

Working With Color, Part II 9

— By Dave Mark



IMPLEMENTOR'S JOURNAL

Implementing Elegant Drag and Drop for Styled Text Fields 21

An implementor's journal of adding drag and drop to SmalltalkAgents®

— By David Simmons, Quasar Knowledge Systems



THINK TOP 10 32

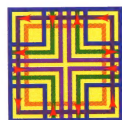
— By Mark B. Baldwin and Michael Hopkins, Symantec Technical Support



PROGRAMMER CHALLENGE

Huffman Decoding 37

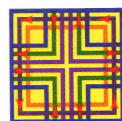
— By Mike Scanlin



ESSENTIAL APPLE TECHNOLOGY

Threading Your Apps 48

Tying it all together — By Randy Thelen, Apple Computer, Inc.



ESSENTIAL APPLE TECHNOLOGY

Thread Manager for Macintosh Applications 56

Apple's Development Guide — By Apple Computer, Inc.



VISUAL PROGRAMMING

Prograph CPX – A Tutorial 69

OOPS! Where'd they put all the semicolons?

— By Kurt Schmucker, Apple Computer, Inc.



INSIDE INFORMATION

Think Like a Moviemaker, Part II 86

You may have a roster that really does look like movie credits...

— By Chris Espinosa, Apple Computer, Inc.



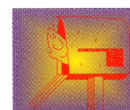
EDITOR'S PAGE

4



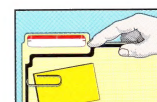
NEWSBITS

81



DIALOG BOX

84



THE CLASSIFIEDS

87



MAIL ORDER STORE

89



**ADVERTISER &
PRODUCT INDEX**

95



TIPS & TIDBITS

96

By Scott T Boyd, Editor



UNCLEAR ON THE CONCEPT

A reader recently sent in an excerpt from the read me for a product which shall remain nameless:

"A common debugging tool many Mac programmers use is called EvenBetterBusError. This tool forces a Mac to crash when certain conditions arise in order to point out POTENTIAL problems to the programmer. The programmer can then assess the situation and decide if there is a REAL problem which needs to be fixed. ProductX happens to frequently create the kinds of situations which EvenBetterBusError watches for. So, if you have EvenBetterBusError installed, you will crash frequently with ProductX – but this is not a bug! Remove EvenBetterBusError to eliminate the crashes. You may be asking yourself why this might concern you – some beta releases of System Software from Apple automatically install EvenBetterBusError when you install them. This little tool may be in your system folder without you knowing it if you are a beta tester for Apple."

First, a little background. EvenBetterBusError was written by Greg Marriott while we were at Apple. Greg was responsible for checking out incompatibility bugs between System 7 (that's 7.0) and third-party software. He became a master bug hunter, and nailed tons of bugs, both in our software and in third-party software. Along the way, he developed several tools for his own detective work, and to give to others so they could have a better chance of catching things before they showed up in his bug list. EvenBetterBusError is a descendant of Mr. Bus Error, which later turned into BetterBusError, and Greg took it even further. Here's what it does.

- It puts a 4-byte value into memory location 0. The value it puts in is designed to generate a bus error or an illegal instruction on any Macintosh. If someone dereferences a nil pointer, they'll immediately generate a bus error. In addition, they'll crash if they jump to 0.

- It periodically checks to see whether someone has written to location 0 (probably using a NIL pointer). If so, it drops into the debugger and says, "Write to NIL."

Both of these behaviors can help you track down misbehaving software early, prior to subjecting your customers to the seemingly-random system "degradation" such bugs can induce.

Here's some reconstituted code which shows you everything that EvenBetterBusError does. I used TMON Pro to disassemble and save the code from the INIT resource, then added comments and labels. As you can see, it's pretty simple. It installs a VBL task which periodically checks location 0, drops into the debugger if it needs to, and stuffs \$50FF8001 back into location 0.

```
BRA.S InstallVBL
BeginCodeBlock
VBLRecord
DC.L 00000000,0001 ; QElemPtr, queue type (1==VBL
queue)
DC.L 00000000 ; ptr to VBL code
```

```
DC.W 0001,0000 ; timeout count & phase count
MOVE.L $0,D0 ; put location 0's contents
into D0
ANDI.L #$7FFFFFFF,D0 ; strip the high bit
CMPI.L #$50FF8001,D0 ; see if someone wrote over
it
BEQ.S SkipDebugStr ; if not, everything's ok
CMPI.L #$40810000,D0 ; see if it's ProcMgr's safe
value
BEQ.S SkipDebugStr ; if so, everything's ok
PEA ItsEvenBetterBusErrorsFault
_DebugStr
SkipDebugStr
MOVE.L #$50FF8001,$0 ; put a bus error value at $0
MOVE.W #$0001,$000A(A0) ; reset the VBL timer
RTS
EndOfCodeBlock

ItsEvenBetterBusErrorsFault DC.B 'Write to NIL',00
SizeOfCodeBlock equ EndOfCodeBlock-BeginCodeBlock

InstallVBL
MOVEQ #SizeOfCodeBlock,D0 ; allocate a block for the
code & data
_NewPtr ,sys ; make a block in the system
heap
MOVE.L A0,-(A7) ; remember this block's
address
MOVEA.L A0,A1
LEA VBLRecord,A0 ; a static copy of our VBL
record
MOVEQ #SizeOfCodeBlock,D0 ; the size of the VBL code &
data
_BlockMove ; copy from INIT rsrc into system
heap block
MOVEA.L (A7)+,A0 ; the system heap block
PEA $000E(A0) ; addr of ptr to VBL in VBL
record
MOVE.L (A7)+,$0006(A0) ; stuff ptr to VBL code
_VInstall
MOVE.L #$50FF8001,$0 ; put a bus error value at $0
RTS
```

You might have noticed my sarcastic label on the string. Every week at Apple, someone unclear on the concept assigned a bug to Greg that read something like, "EvenBetterBusError caused the problem. Taking it out of the System Folder fixed the crashes." This common misperception puts the blame on the messenger, not the culprit. It's *never* ok to write to location 0, nor is it ever ok to dereference a NIL pointer. Code that does these things is *buggy*.

As you might imagine, the above read me file struck me (and Greg) as something of a challenge (really more of an affront, but challenge sounded nicer). In just starting up the software, before even getting to the EvenBetterBusError DebugStr call, the code committed these nefarious acts: it called DisposeHandle on a PICT resource, and called DisposeHandle on an already-disposed handle.

Greg wanted me to call them liars. I'll go him one better. In my opinion, they're boneheads, the kind that give the Macintosh a bad name and get yahoos demanding memory protection in the operating system. On the other hand, kudos to Apple for including EvenBetterBusError in the beta versions!



Don't even think about using a Relational Database System !



Comparing Object Oriented and Relational Design Methodologies

Feature	POET ODBMS	Relational RDBMS
Storing Objects	As Objects	Break Objects into Tables
Database Model	User Application Model	Separate Database Model Required
C++ Integration	Total	Poor
Database Operations	At Object Level	Must Write Code
Productivity	Increased	Reduced
Complex Object Performance	Excellent	Poor

Use the POET Object Database for C++

C++ and class libraries have made the GUI development much easier. After all, it is only logical that more and more developers think in objects. But object orientation shouldn't end at the user interface programming level.

The Problem: Without POET, a C++ programmer must use flat files or a RDBMS to store objects. He has to write code to overcome the mismatch between the application and the database model. This leads to design restrictions, performance penalties and more code to write and maintain.

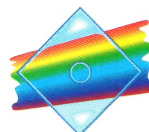
The Solution: POET operates at the object level, it speeds up the development process and provides greater performance. Furthermore, the developer can simply take the object-oriented application design in C++ and map it 1 to 1 into the database. Without any compromise at all!

Full featured database:

POET provides complete support for Encapsulation, Inheritance, Polymorphism and Containers as well as queries, object locking, and transaction handling.

True cross platform support: Complete interoperability makes the development of network enabled applications a breeze. POET supports Macintosh, Power Macintosh, Windows 3.1, Windows NT, Win32s, Windows for Workgroups, Novell Netware (NLM), SUN, AIX, HP-UX, SGI, SCO, OS/2 and NeXTStep.

Call 1-800-950-8845



POET
Software



"Cozzi Ranch relies on our software to get more pigs to market. To fatten up our profits, we rely on the protection leader."

Nothing can eat away profits like a herd of pigs. That's why farmers in 47 countries manage swine production with PigChamp®. And that's why PigChamp protects their revenues with Sentinel™ from Rainbow Technologies.

As a developer, you know the importance of profit margins. If you sell software, you're probably losing revenue to piracy. Protect with Sentinel, and get the revenue you deserve.

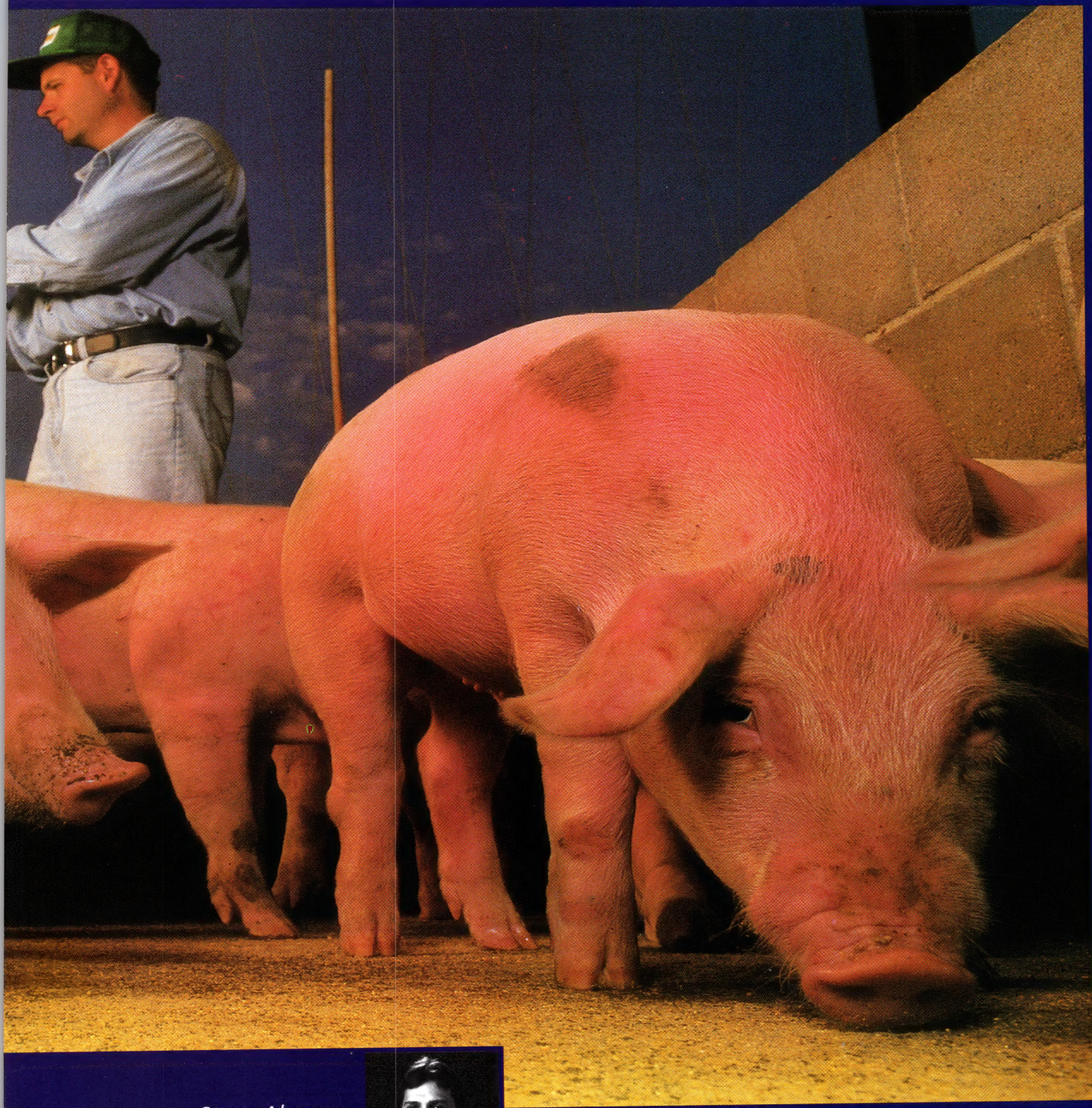


"Our clients produce everything from footballs to kielbasa, and they all get better software at a lower price because we protect with Sentinel.

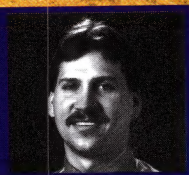
"Illegal duplication can drive the cost of software sky high. Sentinel lets us sell our products inexpensively worldwide, while serving our customers better," explains PigChamp's Steve Abrams.



Rainbow Worldwide HQ: Irvine, CA. Phone: (714) 454-2100 • Fax (714) 454-8557 • AppleLink D3058 • Rainbow U.K. 44 932 570066 • Rainbow France 33 1 47 38 21 21 • Rainbow Germany 49 89 3217 98 0
©1994 Rainbow Technologies, Inc. Sentinel is a trademark of Rainbow Technologies, Inc. PigChamp Software is a trademark of PigCHAMP. All other product names are trademarks of their respective owners. Thanks to Ann and Joe of Cozzi Ranch in Los Baños, California.



Steve Abrams,
PigChamp Software



Over 11,000 developers protect their software with Sentinel. For Macintosh, DOS, Windows, NT, OS/2, UNIX, or any platform – Sentinel is the most advanced protection available. It is the worldwide standard in software protection.

Getting started with Sentinel is quick and easy with a variety of installation options. What's more, Sentinel is truly transparent to your end users. Once installed, they'll never notice it again.

Only Sentinel meets the industry's toughest quality standards and is supported by the industry's largest technical and R&D staff.

So, watch your profits go hog wild. Protect your revenue with confidence. Protect your software with Sentinel.

Order your Sentinel Developer's Kit today!

1-800-852-8569

SENTINEL™
Securing the future of software.



We will not be emulated.

Metrowerks CodeWarrior.
The only native development environment for the Power Macintosh is here.

Bronze	\$ 99
--------	-------

For 68K Mac

Gold	\$399
------	-------

For Power & 68K Mac



Metrowerks CodeWarrior is a screamer.

Experience RISC technology. Compile 200,000 lines a minute on the Power Macintosh 8100.

"Great company, fast compilers... how can you beat that"?

Lee Richardson
Development Manager, MacWrite
Clarisc Corporation

Metrowerks CodeWarrior is awesome.

Native C++ and C compilers for the Power Macintosh. Native C++, C and Pascal compilers for the 68K Macintosh. 68K Macintosh-hosted PowerPC compilers. Power Macintosh and 68K Macintosh source-level debuggers. A new

Power Macintosh and 68K Macintosh application framework—Metrowerks PowerPlant. Tool-Server, SourceServer and other developer utilities from Apple Development Products. It's all here.

"Without the Metrowerks PowerPC compiler it would be virtually impossible to develop Adobe Illustrator for the Macintosh on the PowerPC."

Don Melton
Software Engineering Leader
Adobe Illustrator for Power Macintosh
Adobe Inc.

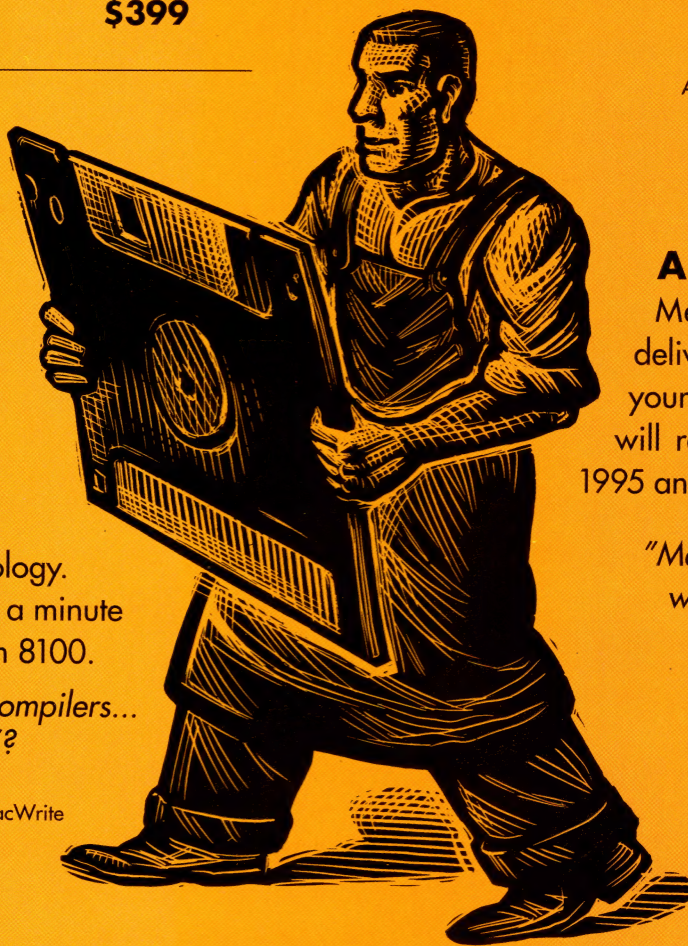
Metrowerks CodeWarrior.

A new way to buy.

Metrowerks CodeWarrior delivers 3 times a year. With your purchase of CW4, you will receive CW5 in January 1995 and CW6 in May 1995.

"Mac developers have been waiting a long time for an environment like Metrowerks CodeWarrior."

Stanley Crane
General Manager R&D,
cc: Mail Division
Lotus Development Corp.



"Call MacTech"
310-575-4343





By Dave Mark, MacTech Magazine Regular Contributing Author

Working With Color, Part 2

Before we get into this month's column, I'd like to take a second to talk about some changes that are coming down the 'pike. Every month, I'm faced with a perplexing challenge. How can I present a decent sized program in my column, and still have room to walk through the source code? The current solution is to split the column over two issues. The first month I present the program's resources and source code and the second month I actually walk through the source code, essentially giving you the source code twice.

Unfortunately, there's no easy way to simplify this process. If I just present the source code as a single block, there's no room for commentary and if I just present the source code with comments mixed in it makes it extremely difficult to type in the code. Not a good situation!

Fortunately, the powers-that-be came up with a pretty cool idea (which I'm sure you'll read about elsewhere in the magazine over the next few months). They are putting together a small framework, called Sprocket, designed to showcase new Mac technologies. Sprocket will start off with same pretty basic capabilities, then grow as time goes on. It will be written entirely in C++ and will be made available to every MacTech subscriber.

This makes life infinitely easier for us column writers! Instead of presenting a single, monolithic application every two months, I'll be able to focus on a few new classes, which can be folded into the framework and, more importantly, which can be presented

and completely discussed, all in a single column.

The biggest implication this has for you is a change from C to C++. If this is a bad thing, now's the time to get it off your chest. Send all bitter, sarcastic complaints to Scott Boyd, at one of the myriad addresses found on page 2 of this magazine.

I'm really looking forward to getting into this new framework and polishing my C++ skills. I think this is a great idea and hope you do too...

BACK TO COLORTUTOR

All that said and done, let's get back to last month's program, ColorTutor. To recap last month's description, ColorTutor is a rewrite of the Mac Primer, Volume II program of the same name. ColorTutor is a hands-on color blending environment. You specify the foreground and background colors and patterns, then select a Color Quickdraw drawing mode. ColorTutor uses `CopyBits()` to mix the foreground and background colors. Figure 1 shows a sample.

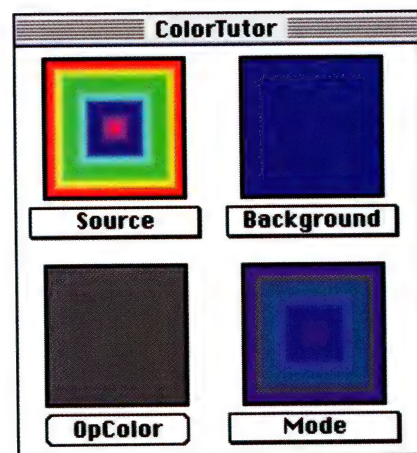


Figure 1. The ColorTutor window.

ColorTutor first copies the Background image to the lower-right rectangle, then copies the Source image on top of the Background using the current Mode and OpColor. As we dig through the code, we'll look at the parts of Color Quickdraw that make ColorTutor possible.

EXPLORING THE COLOR TUTOR SOURCE CODE

ColorTutor starts off with a pair of #includes. <Picker.h> contains the definitions we'll need to use the Mac's built-in color picker, while <GestaltEqu.h> contains what we'll need to call Gestalt().

```
#include <Picker.h>
#include <GestaltEqu.h>
```

Here's the usual cast of constants...

```
#define kBaseResID      128
#define kErrorALRTid    128
#define kNullFilterProc  NULL
#define kMoveToFront    (WindowPtr)-1L
#define kNotNormalMenu  -1
#define kSleep          60L

#define mApple          kBaseResID
#define iAbout          1

#define mFile           kBaseResID+1
#define iQuit           1

#define mColorsPopup    kBaseResID+3
#define iBlackPattern   1
#define iGrayPattern    2
#define iColorRamp      4
#define iGrayRamp       5
#define iSingleColor    6

#define mModePopup      kBaseResID+4
```

We've sure got a lot of globals. Short of adding pass-through parameters to a bunch of routines, I couldn't think of any way to get rid of them. Any ideas?

gDone is the flag that tells us when to drop out of the main event loop. gSrcRect, gBackRect, gDestRect, and gOpColorRect mark the boundaries of the four color rectangles in the ColorTutor window. gSrcMenuRect, gBackMenuRect, and gModeMenuRect mark the outline of the three popup menus. As you read through the code, remember that 'Src' refers to the source color, 'Back' refers to the background color, and 'Op' refers to the op-color (you'll find out what the opcolor is about later in the column).

'Dest' and 'Mode' both refer to the lower-right corner of the ColorTutor window. 'Dest' refers to the lower-right color rectangle which is used to mix the source and background colors. 'Mode' refers to the popup menu below the 'Dest' rectangle and specifies the color drawing mode used to mix the source and background colors.

```
Boolean    gDone;

Rect       gSrcRect, gBackRect, gDestRect, gSrcMenuRect,
           gBackMenuRect, gModeMenuRect, gOpColorRect;
```

The next set of globals mirror the current settings of the three popup menus. gSrcPattern is set to either iBlackPattern or iGrayPattern, and gSrcType is one of iColorRamp, iGrayRamp, or iSingleColor. These choices correspond directly to the menu in Figure 2. gBackPattern and gBackType follow the exact same rules, since the Background menu was built from the same **MENU** resource.

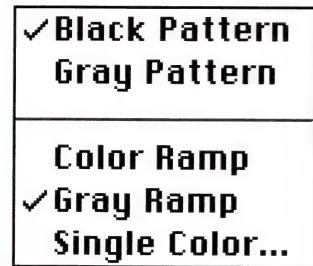


Figure 2. The popup menu that goes along with the Source and Background menus.

gCopyMode is one of the Color Quickdraw copy modes and mirrors the Mode menu shown in Figure 3. gCopyMode is set in the routine UpdateModeMenu().

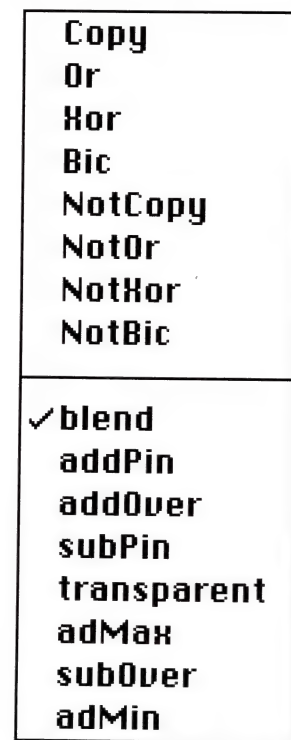


Figure 3. The Mode popup menu.

```
short  gSrcPattern, gBackPattern, gCopyMode, gSrcType, gBackType;
```

gSrcColor, gBackColor, and gOpColor are the colors displayed in the source, background, and op-color rectangles in the ColorTutor window. The source and background colors only come into play when the **Single Color...** item is checked in their respective popup menus. The OpColor is always a single color and comes into play when you use certain Color Quickdraw copying modes.

```
RGBColor gSrcColor, gBackColor, gOpColor;
```

Finally, gSrcMenu, gBackMenu, and gModeMenu are handles

to their respective menus.

MenuHandle gSrcMenu, gBackMenu, gModeMenu;

Here are all the function prototypes...

Functions

```
void ToolboxInit( void );
void MenuBarInit( void );
void CreateWindow( void );
void SetUpGlobals( void );
void EventLoop( void );
void DoEvent( EventRecord *eventPtr );
void HandleMouseDown( EventRecord *eventPtr );
void HandleMenuChoice( long menuChoice );
void HandleAppleChoice( short item );
void HandleFileChoice( short item );
void DoUpdate( WindowPtr window );
void DrawContents( WindowPtr window );
void DrawColorRamp( Rect *rPtr );
void DrawGrayRamp( Rect *rPtr );
void DrawLabel( Rect *boundsPtr, Str255 s );
void DoContent( WindowPtr window, Point globalPoint );
void UpdateSrcMenu( void );
void UpdateBackMenu( void );
void UpdateModeMenu( void );
void DoSrcChoice( short item );
void DoBackChoice( short item );
void DoModeChoice( short item );
short DoPopup( MenuHandle menu, Rect *boundsPtr );
Boolean PickColor( RGBColor *colorPtr );
Boolean HasColorQD( void );
void DoError( Str255 errorString );
```

main() initializes the Toolbox, sets up the menu bar, then checks to see if this machine supports Color Quickdraw. It's important to at least initialize the Toolbox before you check for Color Quickdraw, since we use the Toolbox to report an error if Color Quickdraw is not present.

main

```
void main( void )
{
    ToolboxInit();
    MenuBarInit();

    if ( ! HasColorQD() )
        DoError( "\pThis machine does not support Color QuickDraw!" );
```

Next, we'll create the ColorTutor window, assign initial values to our globals, then drop into the main event loop.

```
CreateWindow();
SetUpGlobals();

EventLoop();
}
```

ToolboxInit() is pretty much the same as always. Notice the use of qd.thePort instead of just plain thePort. THINK C and C++ assume that when you refer to a quickdraw global (like thePort) you are referring to the corresponding quickdraw global that gets allocated as part of each application's memory zone. MPW and CodeWarrior both require the "qd." syntax, turning "thePort" into "qd.thePort". Since THINK C can handle either form, I've gone back to the "qd." form so all my code compiles in either environment.

ToolboxInit

```
void ToolboxInit( void )
```

```
{
    InitGraf( &qd.thePort );
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs( 0L );
    InitCursor();
}
```

MenuBarInit() sets up the menu bar. Notice that it doesn't set up the popup menus, which are not displayed in the menubar.

MenuBarInit

```
void MenuBarInit( void )
{
    Handle menuBar;
    MenuHandle menu;

    menuBar = GetNewMBar( kBaseResID );

    if ( menuBar == NULL )
        DoError( "\pCouldn't load the MBar resource..." );

    SetMenuBar( menuBar );

    menu = GetMHandle( mApple );
    AddResMenu( menu, 'DRVR' );

    DrawMenuBar();
}
```

CreateWindow() creates a new, Color Quickdraw window based on a WIND resource.

CreateWindow

```
void CreateWindow( void )
{
    WindowPtr window;

    window = GetNewCWindow( kBaseResID, NULL, kMoveToFront );

    The call to GetNewControl() uses the CNTL resource
    template to build the OpColor push-button control and add it
    to the window.

    GetNewControl( kBaseResID, window );
```

Finally, the window is made the current port and the port's font is set to Chicago, which is the system font. This is to make sure that the popup menu label is drawn in 12-point Chicago.

```
SetPort( window );
TextFont( systemFont );
}
```

SetUpGlobals() starts off by setting up the four color rectangles and the label rectangles for the three popup menus.

SetUpGlobals

```
void SetUpGlobals( void )
{
    SetRect( &gSrcRect, 15, 6, 95, 86 );
    SetRect( &gBackRect, 125, 6, 205, 86 );
    SetRect( &gDestRect, 125, 122, 205, 202 );
    SetRect( &gOpColorRect, 15, 122, 95, 202 );

    SetRect( &gSrcMenuRect, 7, 90, 103, 108 );
    SetRect( &gBackMenuRect, 117, 90, 213, 108 );
    SetRect( &gModeMenuRect, 117, 206, 213, 224 );
}
```

One way to get rid of these globals is to create a set of rectangle resources, each of which describes a different rectangle. You might create a 'rect' resource type, 8 bytes in

length (4 shorts). Then, instead of using a global, you'd read and write the corresponding 'rect' resource. Even though the globals have a higher overhead, we're talking about a pretty small amount of stack space and I find the resource approach a little cumbersome.

Next, we set default values for our various globals.

```
gSrcPattern = iBlackPattern;
gBackPattern = iBlackPattern;

gCopyMode = srcCopy;

gSrcColor.red = 65535;
gSrcColor.green = gSrcColor.blue = 0;
gSrcType = iSingleColor;

gBackColor.blue = 65535;
gBackColor.red = gBackColor.green = 0;
gBackType = iSingleColor;
```

Here, we create a grey RGBColor (all values are halfway between 0 and 65535) and pass it to OpColor(). OpColor() first checks to make sure the current port is a CGrafPort. If so (and in our case it is), OpColor() sets the rgbOpColor field of the grafvars struct (in the CGrafPort) to the specified color. Take a minute to look at these structures. In *Inside Macintosh: Imaging, Inside Macintosh: Volume V*, or in *THINK Reference*, take a look at the CGrafPort structure. The fourth field is a Handle called grafVars. This is actually a handle to a GrafVars structure.

Now look up GrafVars. The first field, rgbOpColor, contains an RGBColor used with the addPin, subPin, and blend color transfer modes. addPin replaces the destination pixel with the sum of the source and destination pixel colors, up to a maximum of the colors specified by the opColor. With the opcolor as specified below, the red, green, and blue values in the lower-right rectangle of the ColorTutor window will never exceed 32767. With addPin, the maximum color is always white (65535,65535,65535).

```
gOpColor.green = 32767;
gOpColor.red = 32767;
gOpColor.blue = 32767;
OpColor( &gOpColor );
```

subPin replaces the destination pixel with the difference between the source and destination, where the opColor provides a minimum value for the ultimate red, green, and blue fields. With subPin, the minimum color is always black (0,0,0).

Finally, the blend mode uses a weighting formula to blend the source and destination colors:

```
dest = (source * weight / 65535) + (dest * (1-(weight/65535)));
```

This calculation is made once each for red, green, and blue, using the respective opColor field as the weight.

None of the other transfer modes take advantage of a port's opColor.

Next, we'll set up MenuInfo structures for the three popup menus. Each menu is added to the application's menu list by passing its handle to InsertMenu(). The constant

kNotNormalMenu (which is -1L) tells the Menu Manager not to add these menus to the menu bar.

```
gSrcMenu = GetMenu( mColorsPopup );
InsertMenu( gSrcMenu, kNotNormalMenu );

gBackMenu = GetMenu( mColorsPopup );
InsertMenu( gBackMenu, kNotNormalMenu );

gModeMenu = GetMenu( mModePopup );
InsertMenu( gModeMenu, kNotNormalMenu );
}
```

Blah, blah, blah... EventLoop()... blah, blah.

```
void EventLoop( void )
{
    EventRecord  event;

    gDone = false;
    while ( gDone == false )
    {
        if ( WaitNextEvent( everyEvent, &event, kSleep, NULL ) )
            DoEvent( &event );
    }
}
```

DoEvent() does its normal thing, passing mouseDowns to HandleMouseDown() and updates to DoUpdate().

```
void DoEvent( EventRecord *eventPtr )
{
    char  theChar;

    switch( eventPtr->what )
    {
        case mouseDown:
            HandleMouseDown( eventPtr );
            break;
        case keyDown:
        case autoKey:
            theChar = eventPtr->message & charCodeMask;

            if ( (eventPtr->modifiers & cmdKey) != 0 )
                HandleMenuChoice( MenuKey( theChar ) );
            break;
        case updateEvt:
            DoUpdate( (WindowPtr)eventPtr->message );
            break;
    }
}
```

Nothing unusual here. Clicks in the ColorTutor window are handled by DoContent(). Since we only have one window, the call to SelectWindow() will never be made. Oh, well, force of habit...

```
void HandleMouseDown( EventRecord *eventPtr )
{
    WindowPtr  window;
    short      thePart;
    long       menuChoice;

    thePart = FindWindow( eventPtr->where, &window );

    switch ( thePart )
    {
        case inMenuBar:
            menuChoice = MenuSelect( eventPtr->where );
            HandleMenuChoice( menuChoice );
            break;
        case inSysWindow:
            SystemClick( eventPtr, window );
            break;
    }
}
```


Software Developers:

It's Midnight.

Do You Know Where Your Software is?



PowerMAC Compatible!

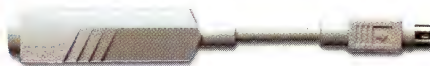
© Aladdin Knowledge Systems Ltd. 1985-1994 (9.94) PowerPC is a trademark of Motorola. Macintosh is a trademark of Apple Inc.

member of



■ Australia Conlab 3 8985685	■ Benelux Aladdin Benelux 080 782098	■ Czech Atlas 2 766085	■ Chile Micrologica 2 222 1388
■ Denmark Berendsen 39 577100	■ Egypt Zeineldein 2 3604632	■ Finland ID-Systems 0 870 3520	■ Germany CSS 201 278804
■ Greece Unibrain 1 6856320	■ Italy Partner Data 2 26147380	■ Japan Athena, 3 58 213284	■ Korea Dae-A 2 848 4481
■ New Zealand Training, 4 5666014	■ Poland Systherm 61 480273	■ Portugal Futurmatica 1 4116269	■ South Africa D Le Roux, 11 886 4704
■ Spain PC Hardware, 3 4493193	■ Switzerland Opag 61 7169222	■ Taiwan Tecco 2 555 9676	■ Turkey Mikrobeta 312 467 7504

Bringing software into the world is a little like bringing up children. You always know where they start, but you seldom know where they'll end up. These days, with illegal use of software so common, concerned developers have good reason to worry about the products of their labor. That's where MachASP comes in.



Like a responsible babysitter, MachASP accompanies your software wherever it goes. With MachASP there, your software won't run out of control. Without MachASP, in fact, your software won't run at all.

For developers, MachASP provides the highest level of security and reliability. For legitimate users, MachASP is a friendly and transparent solution. Once connected, they won't even feel it's there.

And if your child wants to play with its friends, a single Net-MachASP lets it run free around a local area network. But always under your supervision and control.

Get serious about software protection. Since 1984, nearly one million HASP keys have enabled thousands of PC & Mac software developers, in more than 60 countries, to protect their software. To find out why MachASP is considered the best product in the market, order your MachASP Developer's Kit today.

1-800-223-4277

ALADDIN

The Professional's Choice

North America **Aladdin Software Security Inc.**
Tel: (800) 223 4277, 212-564 5678
Fax: 212-564 3377
E-mail: sales@hasp.com

Intl Office **Aladdin Knowledge Systems Ltd.**
Tel: 972-3-537 5795, Fax: 972-3-537 5796
AppleLink: ALADDIN.KNOW
E-mail: aladdin@aladdin.co.il

United Kingdom **Aladdin Knowledge Systems UK Ltd.**
Tel: 0753-622266, Fax: 0753-622262

France **Aladdin France SA**
Tel: 1 40 85 98 85, Fax: 1 41 21 90 56

See us at COMDEX/Fall, Booth S-3554


```

case inContent:
    if ( window != FrontWindow() )
        SelectWindow( window );
    else
        DoContent( window, eventPtr->where );
    break;
case inDrag :
    DragWindow( window, eventPtr->where, &qd.screenBits.bounds );
    break;
}
}

```

HandleMenuChoice(), HandleAppleChoice(), and HandleFileChoice() deserve a column all to themselves, eh?

```

void HandleMenuChoice( long menuChoice )
{
    short menu;
    short item;

    if ( menuChoice != 0 )
    {
        menu = HiWord( menuChoice );
        item = LoWord( menuChoice );

        switch ( menu )
        {
            case mApple:
                HandleAppleChoice( item );
                break;
            case mFile:
                HandleFileChoice( item );
                break;
        }
        HiliteMenu( 0 );
    }
}

```

```

void HandleAppleChoice( short item )
{
    MenuHandle appleMenu;
    Str255 accName;
    short accNumber;

    switch ( item )
    {
        case iAbout:
            SysBeep( 20 );
            break;
        default:
            appleMenu = GetMHandle( mApple );
            GetItem( appleMenu, item, accName );
            accNumber = OpenDeskAcc( accName );
            break;
    }
}

```

```

void HandleFileChoice( short item )
{
    switch ( item )
    {
        case iQuit:
            gDone = true;
            break;
    }
}

```

DoUpdate() handles any update events with calls to DrawContents() to draw everything but the OpColor... push button, which is drawn by calling DrawControls().

```

void DoUpdate( WindowPtr window )
{
    BeginUpdate( window );
}

```

```

DrawContents( window );
DrawControls( window );

EndUpdate( window );
}

```

DrawContents() starts by setting up a true black RGBColor.

```

void DrawContents( WindowPtr window )
{
    RGBColor rgbBlack;
    Rect source, dest;

    rgbBlack.red = rgbBlack.green = rgbBlack.blue = 0;
}

```

Next, the CGrafPort's pattern is set to black or gray, depending on the value of gSrcPattern, which mirrors the setting of the Source popup menu. Here's another place I could have gotten rid of some globals. Instead of checking the value of a global, I could have checked the appropriate menu item to see if it was checked. Again, I find the global-based method to be less cumbersome, though there are some who, even as we speak, are reaching for their direct lines to the Thought Police in their desperate quest for interface cleanliness. Sigh.

```

if ( gSrcPattern == iBlackPattern )
    PenPat( &qd.black );
else
    PenPat( &qd.gray );

```

Next, we'll draw either a color ramp, a gray ramp, or a solid color in the source rectangle. If you don't know what a ramp is, check out the source rectangle in Figure 1.

```

if ( gSrcType == iColorRamp )
    DrawColorRamp( &gSrcRect );
else if ( gSrcType == iGrayRamp )
    DrawGrayRamp( &gSrcRect );
else
{
    RGBForeColor( &gSrcColor );
    PaintRect( &gSrcRect );
}

```

Now we'll repeat this process for the background rectangle.

```

if ( gBackPattern == iBlackPattern )
    PenPat( &qd.black );
else
    PenPat( &qd.gray );

if ( gBackType == iColorRamp )
    DrawColorRamp( &gBackRect );
else if ( gBackType == iGrayRamp )
    DrawGrayRamp( &gBackRect );
else
{
    RGBForeColor( &gBackColor );
    PaintRect( &gBackRect );
}

```

Next, we'll restore the pen pattern to solid black, paint the opColor rectangle, then set the ForeColor() back to black.

```

PenPat( &qd.black );

RGBForeColor( &gOpColor );
PaintRect( &gOpColorRect );

RGBForeColor( &rgbBlack );

```

Next, we'll draw the three popup menu labels, then the frames around the four color rectangles, then return the pen to normal.



Incredibly Easy To Use

Our toolkits are so easy to use that it should take you less than an hour to add complete image import, export, conversion, display, printing and scanning support to your application. If you need to work at a lower level for special situations, we have functions for that, too!

Story Behind The Guarantee

Having offered this Guarantee for over two years, we have collected several thousand (weird, but perfectly valid) images. Nowadays we see very few problem images. If we do, the image is usually corrupted or invalid, yet we are able to provide solutions for these images as well.

Performance Tuned By Experts

We at AccuSoft have an unyielding commitment to achieving the highest performance possible for all our products. This, of course, means that your products will also achieve greater performance.

Support For All Platforms

AccuSoft has versions for all major platforms including DOS, 32-bit DOS, Clipper™, Foxpro™, Windows™, Visual Basic®, Windows NT, OS/2, Chicago, Macintosh®, and UNIX. We can also port to other systems upon request.

Complete Compression

All forms of compression are supported including JPEG, Group III, Group IV, Packbits, LZW, Huffman, and more. Read any image format, then convert to any other format-with only two function calls!

AccuSoft is the Fastest!



Now get the fastest imaging toolkit with the most platforms, the most formats, and the only guarantee.

AccuSoft is now the recognized provider of the highest quality imaging toolkits in the world. Our performance, compatibility, ease-of-use, service, and our AccuSoft Image Guarantee have earned us this distinction. Our libraries save you hundreds of hours of work and provide your application with a unique advantage: Guaranteed Format Support.

Features, Features, Features

We provide a complete set of functions for printing, scanning, display, image processing and image handling. All printers and TWAIN scanners are supported with simple function calls. The image printing engine produces high quality output regardless of the printer. The display engine provides high-speed (up to 50 times faster than Windows) and high quality display for all types of images and display modes. Our image processing functions include zoom, pan, scroll, invert, rotate, resize, sharpen, blur, contrast, brightness, palette optimization, color reduction, matrix convolutions and much more.

Visual Basic® Versions

We have special versions for Visual Basic that provide all the power of the DLL in the form of a Custom Control. We even have the world's only 32-bit VBX for Windows 3.1. *Extremely Fast!*

Pro Gold Versions

Our Pro Gold libraries represent the most advanced performance and features available in the world. Pro Gold versions are available for Windows 3.1 (no special hardware or drivers required), Mac, UNIX and others. When your application demands unbeatable performance, go with Pro Gold.

Call Now To Order 800-525-3577

Risk-Free 30-day money back guarantee on all 16-bit products.



**Guaranteed to read all raster images in existence in the supported formats. If you can find a valid image we don't read, send it to us and we will make it work.*



©Copyright 1994 AccuSoft Corporation. All Rights Reserved.

AccuSoft Corporation, 112 Turnpike Road, Westborough, MA 01581 TEL:(508) 898-2770 FAX: (508) 898-9662

```
DrawLabel( &gSrcMenuRect, "\pSource" );
DrawLabel( &gBackMenuRect, "\pBackground" );
DrawLabel( &gModeMenuRect, "\pMode" );
```

```
PenSize( 2, 2 );
FrameRect( &gSrcRect );
FrameRect( &gBackRect );
FrameRect( &gDestRect );
FrameRect( &gOpColorRect );
```

```
PenNormal();
```

Now we'll set up the source and destination rectangles. We inset them to avoid copying the frames. Notice that we first copy the Background rectangle to the Dest (lower-right) rectangle. Once we do that, we'll copy the Source rectangle on top of that.

```
source = gBackRect;
InsetRect( &source, 2, 2 );
```

```
dest = gDestRect;
InsetRect( &dest, 2, 2 );
```

Here's our call to CopyBits(). Since this first call is just intended to get the background pixels down to the lower-right rectangle, we'll use the srcCopy transfer mode.

```
CopyBits( (BitMap *)&(((CGrafPtr>window)->portPixMap),
          (BitMap *)&(((CGrafPtr>window)->portPixMap),
          &source, &dest, srcCopy, NULL );
```

Now we'll copy the Source rectangle down to the lower-right rectangle, overwriting what we just copied with the previous call to CopyBits(). The results will change, depending on the transfer mode in gCopyMode. The transfer modes are described in *Inside Macintosh: Volume V*, *Inside Macintosh: Imaging*, and in *THINK Reference*, but I think the best description by far is in *IM: Imaging*. If you plan on working with color to any great extent, pick up *IM: Imaging* and check out *IM: Advanced Imaging* (not for most folks).

```
source = gSrcRect;
InsetRect( &source, 2, 2 );

CopyBits( (BitMap *)&(((CGrafPtr>window)->portPixMap),
          (BitMap *)&(((CGrafPtr>window)->portPixMap),
          &source, &dest, gCopyMode, NULL );
```

The color ramp stuff is kind of interesting. Instead of using the red, green, blue color model, we depend on the hue, saturation, and brightness model. We vary the hue from 0 to 65535 with the resolution determined by width, in pixels, of the specified rectangle.

```
void DrawColorRamp( Rect *rPtr )
{
    long numColors, i;
    HSVColor hsvColor;
```

DrawColorRamp


```

RGBColor rgbColor;
Rect r;

r = *rPtr;

```

We inset the rectangle by 2 pixels so we don't draw on the rectangle's frame. Interestingly, hue is hue, saturation is saturation, but for some reason, brightness is represented by the field named value. Hmm... Oh, well. As you can see, our color ramp will consist of bright, saturated colors. Try rewriting the code to ramp on saturation or brightness instead.

```

InsetRect( &r, 2, 2 );
numColors = ( rPtr->right - rPtr->left - 2 ) / 2;
hsvColor.value = hsvColor.saturation = 65535;

```

Here's the ramping loop. HSV2RGB() converts an HSV color to an RGB color, which produces something we can pass to RGBForeColor(). Why isn't there an HSVForeColor() routine? I don't know, but I wish there were.

```

for ( i = 0; i < numColors; i++ )
{
    hsvColor.hue = i * 65535 / numColors;
    HSV2RGB( &hsvColor, &rgbColor );
    RGBForeColor( &rgbColor );
}

```

Each time through the loop, we draw a 1 pixel wide rectangle, then shrink the rectangle in preparation for the next pass through the loop.

```

    FrameRect( &r );
    InsetRect( &r, 1, 1 );
}

```

DrawGrayRamp() does essentially the same thing, but stays within the RGB model.

```

void DrawGrayRamp( Rect *rPtr )
{
    long numColors, i;
    RGBColor rgbColor;
    Rect r;

    r = *rPtr;
    InsetRect( &r, 2, 2 );
    numColors = ( rPtr->right - rPtr->left - 2 ) / 2;
}

```

In this case, we run red from 0 to 65535, with a step resolution based on the width of the rectangle. We then copy the red value into both green and blue, producing progressively lighter shades of gray.

```

for ( i = 0; i < numColors; i++ )
{
    rgbColor.red = i * 65535 / numColors;
    rgbColor.green = rgbColor.red;
    rgbColor.blue = rgbColor.red;

    RGBForeColor( &rgbColor );

    FrameRect( &r );
    InsetRect( &r, 1, 1 );
}

```

In real life, this routine shouldn't be necessary. We really should be using the popup menu CDEF that's been around for several years. The problem is, ResEdit has a bug in it that

makes the CDEF very difficult to set up properly. I'm not sure if the problem lies with ResEdit or with the CDEF, but just to avoid the problem, here's the old way of doing popups. What this really needs is that nifty down-pointing triangles that you usually see in popups, but I ran out of time. Aside from spending the money for Resorcerer (a good investment, IMHO), does anyone have a workaround that brings the popup CDEF under control (sorry!!) in ResEdit?

```

void DrawLabel( Rect *boundsPtr, Str255 s )
{
    Rect r;
    int size;
}

```

This code basically draws a poor imitation of a popup menu label, using StringWidth() to calculate the proper centering position in the label.

```

r = *boundsPtr;
r.bottom -= 1;
r.right -= 1;
FrameRect( &r );

MoveTo( r.left + 1, r.bottom );
LineTo( r.right, r.bottom );
LineTo( r.right, r.top + 1 );

size = boundsPtr->right - boundsPtr->left - StringWidth(s);
MoveTo( boundsPtr->left + size / 2, boundsPtr->bottom - 6 );

DrawString( s );
}

```

When we get a click in the content region of the window, DoContent() takes a global mouse position and converts it to the local coordinate system. I probably should have added a SetPort() call to make sure that window was the current window before I called GlobalToLocal(). Since there's only one window, this won't cause any harm here, but be sure to add the SetPort() if you reuse this code.

```

void DoContent( WindowPtr window, Point globalPoint )
{
    int choice;
    ControlHandle control;
    RGBColor rgbColor;
    Point p;

    p = globalPoint;
    GlobalToLocal( &p );
}

```

If the click was in a control, we'll call TrackControl() to track the mouse till the button is released.

```

if ( FindControl( p, window, &control ) )
{
    if ( TrackControl( control, p, NULL ) )
    {

```

If the mouse was released inside the control, we know it was in the OpColor... push-button. We'll call PickColor() to put up the standard Mac color picker.

```

        rgbColor = gOpColor;
        .if ( PickColor( &rgbColor ) )

```

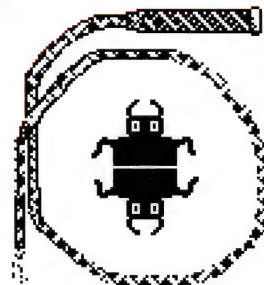

Gives you the Information to Program your Best!



Information

The Debugger V2 & MacNosy

by Steve Jasik



Control

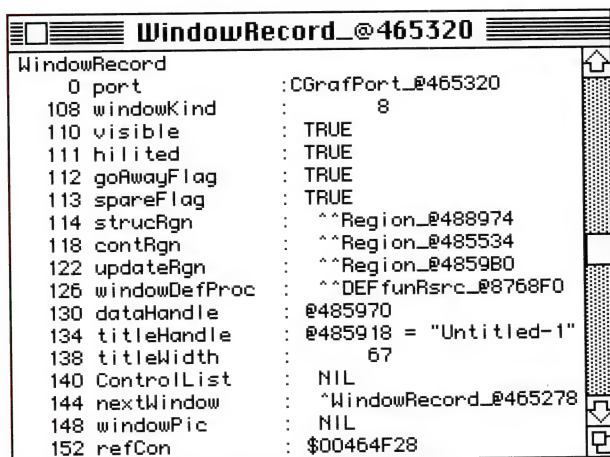
The Debugger is a low and high-level symbolic Debugger that runs in a full multi-window Macintosh environment. You can trace program execution, view the values of variables, etc. **of both 68K and PowerPC programs.**

MacNosy is a global interactive disassembler that enables one to recover the source code of any Mac application, resource file or the ROM.

When you compare features of the different debuggers, note that *only one* has all the below features to help you get your job done, and *only one* has MacNosy to help you debug any program in a full system (6.0x or System 7.x) environment symbolically!

It is the *only* debugger to use the MMU to protect your CODE resources and the rest of the system from the program you are debugging. With MMU Protection you can find errors when they happen, not millions of instructions later! (Macintoshes with 68030 CPUs only).

The Debugger is the debugger of choice at: Adobe, Aldus, Claris, Electronic Arts, Kodak, Metrowerks, etc.



An example of a structured data display window

Its Features Include:

- **Symbolic Debugging** of any Macintosh program, ROM, or code resource (DRVRs, XCMDs, INITs, PDEFs, 4DEXs ..)
- **Source level debugging for Metrowerks & MPW** compiled programs (C++, C, Pascal, Fortran, ...), and an Incremental Build System with instant Link for superfast development.
- **Object Inspector for MacApp 3 programs**
- **Source level debugging of Think C™ projects**
- **Includes a program (CoverTest) to interactively do Code Coverage analysis for SQA testing, etc.**
- **Simultaneous Symbolic debugging of multiple "tasks"**
- **Fast Software Watchpoint** command to find clobbered variables
- **Sophisticated error check algorithms** such as Trap Discipline (Argument Checking), Handle Zapping, Heap Scramble and Heap Check to detect program errors before they become disasters
- **Structured display** of data (hypertext) with user definable structures while debugging
- **Conditional breakpoints** to help filter out redundant information
- **Continuous Animated Step Mode** to watch your program execute instruction by instruction
- **Detailed symbolic disassembly for both 680x0 and PowerPC** with symbol names, labels, cross ref maps, - make it possible to ferret out the secrets of the ROM, etc.
- **"Training Wheels"** for the PowerPC disassembler to help you learn the opcodes

The Debugger V2 & MacNosy: \$350

Runs on all Macs. Call For Group prices or Updates.
Visa/MC Accepted.

Available from: Jasik, APDA, Frameworks or ComputerWare (800-326-0092).

Jasik Designs • 343 Trenton Way, Menlo Park, California 94025 • (415) 322-1386

Internet: macnosy@netcom.com • Applelink: D1037


```
{
```

If the user clicked the OK button to pick a new opColor, we'll force an update and set the new opColor.

```
    gOpColor = rgbColor;
    InvalRect( &gOpColorRect );
    InvalRect( &gDestRect );
    OpColor( gOpColor );
}
}
```

If the click was in the source menu, we'll update the check marks, then call DoPopup() to popup the menu. If an item is chosen from the menu, we'll pass the item to DoSrcChoice(), then force an update.

```
else if ( PtInRect( p, &gSrcMenuRect ) )
{
    UpdateSrcMenu();

    choice = DoPopup( gSrcMenu, &gSrcMenuRect );

    if ( choice > 0 )
    {
        DoSrcChoice( choice );

        InvalRect( &gSrcRect );
        InvalRect( &gDestRect );
    }
}
```

If the click was in the background or mode menus, we'll follow the same plan, with calls to DoBackChoice() or DoModeChoice() as a result.

```
else if ( PtInRect( p, &gBackMenuRect ) )
{
    UpdateBackMenu();

    choice = DoPopup( gBackMenu, &gBackMenuRect );

    if ( choice > 0 )
    {
        DoBackChoice( choice );

        InvalRect( &gBackRect );
        InvalRect( &gDestRect );
    }
}
else if ( PtInRect( p, &gModeMenuRect ) )
{
    UpdateModeMenu();

    choice = DoPopup( gModeMenu, &gModeMenuRect );

    if ( choice > 0 )
    {
        DoModeChoice( choice );

        InvalRect( &gDestRect );
    }
}
}
```

UpdateSrcMenu() starts by unchecking all its items.

```
void UpdateSrcMenu( void )
{
    int i;

    for ( i = 1; i <= 6; i++ )
```

```
    CheckItem( gSrcMenu, i, false );
```

Next, it uses globals to decide which items should be checked.

```
    if ( gSrcPattern == iBlackPattern )
        CheckItem( gSrcMenu, iBlackPattern, true );
    else
        CheckItem( gSrcMenu, iGrayPattern, true );

    if ( gSrcType == iColorRamp )
        CheckItem( gSrcMenu, iColorRamp, true );
    else if ( gSrcType == iGrayRamp )
        CheckItem( gSrcMenu, iGrayRamp, true );
    else if ( gSrcType == iSingleColor )
        CheckItem( gSrcMenu, iSingleColor, true );
}
```

UpdateBackMenu() does the same thing as UpdateSrcMenu().

```
void UpdateBackMenu( void )
{
    int i;

    for ( i = 1; i <= 6; i++ )
        CheckItem( gBackMenu, i, false );

    if ( gBackPattern == iBlackPattern )
        CheckItem( gBackMenu, iBlackPattern, true );
    else
        CheckItem( gBackMenu, iGrayPattern, true );

    if ( gBackType == iColorRamp )
        CheckItem( gBackMenu, iColorRamp, true );
    else if ( gBackType == iGrayRamp )
        CheckItem( gBackMenu, iGrayRamp, true );
    else if ( gBackType == iSingleColor )
        CheckItem( gBackMenu, iSingleColor, true );
}
```

UpdateModeMenu() starts off the same way, by unchecking the mode menu.

```
void UpdateModeMenu( void )
{
    int i;

    for ( i = 1; i <= 17; i++ )
        CheckItem( gModeMenu, i, false );
```

To understand this next chunk of code, take a look at the definition of the transfer modes in <Quickdraw.h>. The first 8 transfer modes are defined by an enum as 0 through 7 and correspond to items 1 through 8 in the Mode menu. The next 8 transfer modes are enum'ed as 32 through 39.

```
    if ( ( gCopyMode >= 0 ) && ( gCopyMode <= 7 ) )
        CheckItem( gModeMenu, gCopyMode + 1, true );
    else
        CheckItem( gModeMenu, gCopyMode - 22, true );
}
```

DoSrcChoice() and DoBackChoice() use their choices to update their globals. If Single Color... is selected, PickColor() is called to select a new color.

```
void DoSrcChoice( short item )
{
    RGBColor rgbColor;

    switch ( item )
```



```

{
    case iBlackPattern:
    case iGrayPattern:
        gSrcPattern = item;
        break;
    case iColorRamp:
    case iGrayRamp:
        gSrcType = item;
        break;
    case iSingleColor:
        gSrcType = iSingleColor;
        rgbColor = gSrcColor;

        if ( PickColor( &rgbColor ) )
            gSrcColor = rgbColor;
        break;
}
}

```

DoBackChoice

```

void DoBackChoice( short item )
{
    RGBColor rgbColor;

    switch ( item )
    {
        case iBlackPattern:
        case iGrayPattern:
            gBackPattern = item;
            break;
        case iColorRamp:
        case iGrayRamp:
            gBackType = item;
            break;
        case iSingleColor:
            gBackType = iSingleColor;
            rgbColor = gBackColor;

            if ( PickColor( &rgbColor ) )
                gBackColor = rgbColor;
            break;
    }
}

```

DoModeChoice() uses its selection to update gCopyMode.

DoModeChoice

```

void DoModeChoice( short item )
{
    if ( ( item >= 1 ) && ( item <= 8 ) )
        gCopyMode = item - 1;
    else
        gCopyMode = item + 22;
}

```

DoPopup() calls PopUpMenuSelect() to implement a popup menu.

DoPopup

```

short DoPopup( MenuHandle menu, Rect *boundsPtr )
{
    Point corner;
    long theChoice = 0L;

    corner.h = boundsPtr->left;
    corner.v = boundsPtr->bottom;

    LocalToGlobal( &corner );

    InvertRect( boundsPtr );

    theChoice = PopUpMenuSelect( menu, corner.v - 1, corner.h + 1, 0);

    InvertRect( boundsPtr );

    return( LoWord( theChoice ) );
}

```

PickColor() is just a wrapper for GetColor(). It sets up a Point with coordinates (-1, -1), which tells the Color Quickdraw routine GetColor() to center the Color Picker on the main screen.

PickColor

```

Boolean PickColor( RGBColor *colorPtr )
{
    Point where;

    where.h = -1;
    where.v = -1;

    return( GetColor( where, "\pChoose a color...", colorPtr,
        colorPtr ) );
}

```

HasColorQD() calls Gestalt() to see if Color Quickdraw is installed on this machine. If the third byte of the returned long is positive, Color Quickdraw is installed.

HasColorQD

```

Boolean HasColorQD( void )
{
    unsigned char version[ 4 ];
    OSErr err;

    err = Gestalt( gestaltQuickdrawVersion, (long *)version );

    if ( version[ 2 ] > 0 )
        return( true );
    else
        return( false );
}

```

DoError() is our standard error handling routine.

DoError

```

void DoError( Str255 errorString )
{
    ParamText( errorString, "\p", "\p", "\p" );

    StopAlert( kErrorALRTid, kNullFilterProc );

    ExitToShell();
}

```

TILL NEXT MONTH

Try experimenting with the code. I especially like messing with different color and grayscale ramps. Try writing a program that blends two offscreen pixmaps on screen using different drawing modes. Try modifying ColorTutor so, as the cursor passes over a pixel, the RGB and HSV values are displayed. Change ColorTutor so it uses the popup CDEF instead of the old-style popups.



**To receive information on any products
advertised in this issue, send your request
via Internet:
productinfo@xplain.com**

FREE INSTALLER!

To: Mac Developers & Product Managers

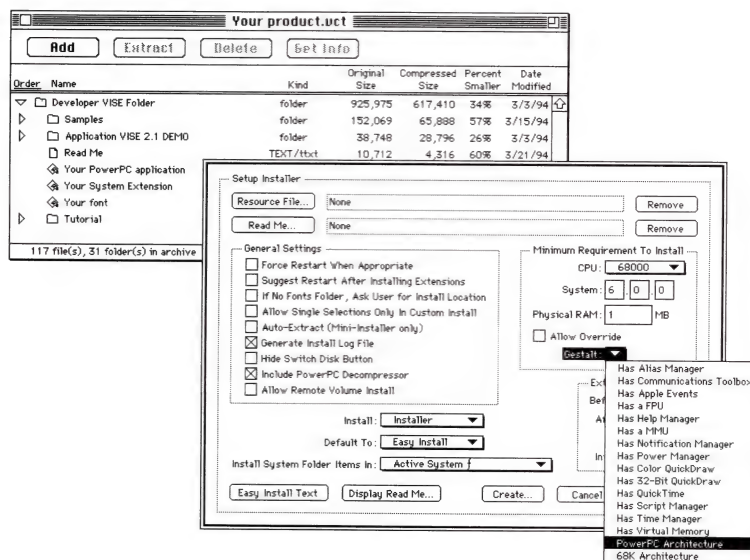
From: MindVision (Creators of Stacker for Macintosh)

Subject: Free Copy of DEVELOPER VISE 3.0

**Message: We've got a great new installer for you.
Here's your chance to try it for free.
No risk, no obligation, no hassle.
Call today, don't delay!**

**Full PowerPC
Support**

**Integrated
compression**



**No programming
required**

**Fully graphical
interface**

Call (402) 477-3269

IF YOU PREFER, USE E-MAIL. APPLELINK, AOL: MINDVISION • COMPUSEVER 70253,1437

Join the growing list of companies who use our VISE technology: Adobe, Apple, CE Software, Claris, CompuServe, DeltaPoint, MacroMedia, Radius, Stac Electronics, Symantec, WordPerfect, and many more.

MindVision Software 840 South 30th St., Suite C, Lincoln, Nebraska 68510
Voice: (402) 477-3269 Fax: (402) 477-1395 AppleLink, AOL: MindVision

© 1992-94 MindVision Software. All Rights Reserved. Developer VISE is a trademark of MindVision Software.





By David Simmons, Quasar Knowledge Systems

Implementing Elegant Drag and Drop for Styled Text Fields

An implementor's journal of adding drag and drop to SmalltalkAgents®

Utilizing the drag manager effectively in your applications can be difficult, but it doesn't have to be. In this article I will be providing a reconstructed journal of my efforts in putting in drag and drop support into SmalltalkAgents styled text field components (<TEField> class). However, before we jump into the code and annotated explanations, I am going to provide a little background information to set the stage.

The design of SmalltalkAgents direct manipulation interface for the Agents Object System (AOS) Presentation Manager (APM) View System required that we architect specific drag and drop functionality into existing components. One of the more challenging designs was that of the styled text field component. Apple's sample applications that came with their Drag and Drop Manager toolkit are quite well done and have a number of subtleties. We wanted to be sure that our Smalltalk implementation was up to the Macintosh user-interface standards.

SmalltalkAgents' Drag and Drop Architecture was derived by examining a large number of applications existing on different operating systems. We wanted to understand how they handled various

drag and drop operations including text handling. We also examined the protocols and frameworks for X/Motif, MS-Windows, OpenDoc, Apple Drag and Drop, and Taligent's architecture to ensure that our framework was generic enough that we could transparently plug into host system frameworks when they were available; and when they were not, we wanted SmalltalkAgents internal framework to have a full-featured architecture of its own.

To accomplish these goals we felt that we had to provide a complete internal Smalltalk architecture that could stand on its own as well as transparently integrating with a given host system. This meant that if a given host system was missing any features our architecture would dynamically supply the missing functionality.

Throughout the remainder of this article I will usually refer to various generic features of SmalltalkAgents framework, the first time I do so, I will try to present the equivalent Apple Drag Manager issues as well as explanations for non-Smalltalk programmers.

BACKGROUND

As part of preparing the reader, we need to cover some of the basics of what a component performing drag operations is responsible for. Drag operations can be broken down into three distinct areas. First is the initiation of a drag operation, second is the tracking and visual drag acceptance feedback, and third is the drop handling.

Drag initiation for text fields involves modification of the button-press handlers of text components. Specifically it requires that an additional test be made on a button-press to determine if the click occurred inside a selection range of one or more characters. If the test was true then a drag package must be built and the drag manager invoked to handle the dragging and possible dropping.

Drag tracking involves three subsidiary operations for handling a drag action. The first is the drag-enter test where the text-field determines if it has any interest in the drag-package. If it does, then it usually is necessary to allocate scratch data for the tracking within operation. Assuming that it does, the second

David Simmons – David works for Quasar Knowledge Systems, Inc., and is the Chief Architect of SmalltalkAgents®.

phase involves tracking the drag operation while the cursor is inside the text field (i.e., drag-within hiliting and auto-scrolling). The third phase is the cleanup operation that the text-field must perform when the cursor exits the text field (i.e., the drag-exit de-hiliting and release of any allocated scratch data).

The drop handling operation is based on a callback to the component (in our case, a text-field) in which the drop occurred. The drop operation involves a number of distinct steps: the first step is to determine if the operation was a self move, or a copy operation. If it was a copy operation then it is simple. If it was a self move operation then we need to handle the animation of the text move within the text field.

AND NOW THE CODE!

As a starting point I began the work in the drag tracking methods <TEField> inherited from the <UIComponent> class. As a point of reference, in Smalltalk terminology a "method" is equivalent to the C term "function". My first task was the filtering of drag requests so that I could determine whether or not to display any form of visible user-feedback to indicate whether or not the dragged package of items contained something of interest for my text field. To do that I needed to specialize the <#dragEnter> method in the <TEField> class.

TEField[i]dragEnter

```
"Validate whether the package is interesting"
(Mouse dragPackageDetect:
 [:type | (type isKindOf: String class)])
 ifFalse: [^self].
```

In the above code block we are using <#dragPackageDetect:> method to iterate over the contents of the drag package and inject the "type-of-object" that would be instantiated if we are to ask for the drag package contents (or some subportion of the contents).

If nothing interesting is in the drag package then we simply exit via "self".

Note: The drag package is a "promise" of what will be delivered and that in most cases the actual data is not instantiated (reified or made-real) until you ask for it.

```
"Compute the drag area and then display it"
outer := self contentBounds asRegion copy insetBy: (-1 @ -1).
inner := outer copy insetBy: 2.
outer differenceWith: inner.
```

Here we are computing the content region of our text-field (<TEField> instance) and then creating a 2 pixel wide region that surrounds it.

```
Mouse showDragHiliteRegion: outer.
```

Now tell the Smalltalk drag-manager object (i.e., the <Mouse>; an instance of <CursorDevice>) to display the hilite region. We must ask our drag manager to display the hilite area because it is managing the display of the drag-image and we want to be assured that our hilite region is drawn "visually-behind" the drag-image.

```
"Indicate that we are tracking the drag operation, but that
it needs initialization."
Mouse dragTargetData: false.
```

Finally since the package was interesting we set the active-drag-target's reference object to be non-nil. We can use this for anything we want as long as we are the active drag target. When we lose targeting the reference value will be cleared. Since we own this value (for now) we set it to <false> to indicate that it is interesting but that we are not yet initialized. We will see more about this when we get to the <#dragWithin> and <#dragExit> methods.

So, we have accomplished our first goal. We now can

detect a package of interesting information and hilite ourselves as appropriate. What we need to do next is make sure that when we lose drag-targeting we will be un-hilited and that any temporary data structures we created are properly de-referenced (i.e., similar to the notion of destruction in C++).

In Smalltalk the developer does not need to track objects they are using; the virtual-machine tracks and disposes of any objects that are not referenced anywhere in the known object-spaces. To be considered as "referenced", an object must be referenced through a chain of object references that leads back to a persistent (non-garbage-collected/anchored) object.

So, our only job is to make sure that we do not have any "anchored" objects referencing the temporary objects we created. Again, this is not really necessary but it is more efficient for us to aid the automatic garbage collector by providing it with hints so that it can dispose of these objects more quickly because there may be a large graphic or some other memory intensive object hanging around from our hiliting-feedback operations.

To accomplish our goal we must specialize the <#dragExit> method which <TEField> inherited from the <UIComponent> class.

TEField[i]dragExit

```
(data := Mouse dragTargetData) ifTrue:
 [
  "Hide any visible blinking bar we may have showing"
  (data at: 6) value: false.
 ]
```

If there was any dragging-action that took place within the <TEField> target then we may have created some temporary structures. If so, then make sure that we invoke a good-bye kiss to clean-up. The use of "(data at: 6)" is based on our knowledge of what we created in our TEField[i]dragWithin method.

Note: We can still write this code and test our operations even though we haven't written the <#dragWithin> method yet.

```
^super dragExit
```

The above statement invokes the standard <#dragExit> behavior that was inherited from our superclasses. The default method and/or the <Mouse> will take care of clearing the tracking-feedback hiliting and clearing the drag-target-data.

By the way, I should point out two things now. First, we are guaranteed by the SmalltalkAgents' Drag and Drop Architecture that a component will always be sent the following sequence of messages during a drag tracking operation:

```
#dragEnter
#dragWithin "1 or more times depending on whether the
             Mouse moves inside of the field"
#dragExit
```

Second, as I mentioned at the beginning, this article is essentially a re-construction of the interactive sequence of coding that I performed while building the text-field drag code. The reason why this is relevant now is because, by this stage, I had live Drag and Drop hiliting feedback in all the <TEField> instances of my Smalltalk modules (a module is an application that shares the Smalltalk environment in a DLL/CFM-like fashion). So far, I was about 5-10 minutes into my development time.

Finally, to complete our tracking we need to provide cursor insertion point feedback as we track the dragging of an a

“interesting” package within our TEFIELD instance. So, as you might expect by now, we need to specialize the <#dragWithin> instance method in TEFIELD.

In Smalltalk, instance methods are the behavior that “instances” of a given class will exhibit. Class methods are the behavior that a given class itself will exhibit. Classes are first-class objects which means that they are real and can be sent message just like any other object. In fact, classes are instances of Metaclass or one of its subclasses.

Before proceeding into our design of the <#dragWithin> method, let me point out a bit about my design intentions. I am going to create all the temporary drag operation structures inside the <#dragWithin> method to guarantee locality of reference. By doing so, it will be easier (for a person) to understand what was happening and thus it will be easier to maintain the code (i.e., we encapsulate most of the drag-action behavior inside the <#dragWithin> method).

We can accomplish this because SmalltalkAgents, and many other Smalltalk’s (Digitalk Smalltalk/V is one exception), support closures (like in Lisp). In Smalltalk, a closure is constructed using a block declaration, which is part of the basic grammar and semantics of the Smalltalk language.

A block declaration is like a nameless function (or ProcPtr), that when it is first referenced, will result in a <BlockClosure> being instantiated. The resulting <BlockClosure> will also retain the contextual information of the “context” (also known as its “execution-world”) in which it was instantiated. Specifically, it will retain a reference to all the shared method temporary (local) variables that we allocated while the enclosing method’s call frame is active and it will also include any shared block temporary variables defined in any (enclosing) outer-blocks. Blocks are a fundamental part of the Smalltalk language and fortunately it only takes a little bit of time to understand them and learn how to use them effectively.

As contexts are created (reified) for shared use by <BlockClosure> instances, the virtual-machine will relocate any of our temporary stack variables that are now shared, and it will update our call-frame to indicate where to find the variables whose “context” is being shared between the blocks and the compiled-method’s call frame in which the blocks were instantiated. This is necessary because in Smalltalk, blocks are first class objects (as are methods by the way) and can be assigned, passed around, and sent messages to. Specifically, we can create a block within a method and then evaluate that block after the method has returned (i.e., after it has exited). Should that happen, the block(s) need to be ensured that they can still access the same variables (scope/context) that existed when the blocks were created.

Ok, let’s look at what we need to do in our <#dragWithin> method.

TEFIELD[i]dragWithin

| bottom offset point line localPosition height top data |

The above construct is the list of method temporaries (i.e., local variables) that need to be allocated when the method is invoked. The actual declaration of the local variables is optional because the compiler automatically detects variable usage as it processes

the code. The SmalltalkAgents browser tools use this capability to provide “authors” (developers) with the option to automatically have the declarations pasted into their source code for them.

Note that in SmalltalkAgents the allocation of temporary variables is initially allocated on the active thread’s stack (SmalltalkAgents is pre-emptively multi-threaded). The temporary variables are also initialized by Smalltalk to point to <nil>, which is the sole instance of the <UndefinedObject> class.

```
data := (Mouse dragTargetData) ? [^self].
```

The first action our method takes is to check and see if there was anything interesting in the drag package. Remember that in the <#dragEnter> method, if the drag-package was interesting we set the <dragTargetData> to be <false>. If it was not interesting, then we left the <dragTargetData> as <nil>.

As an aside, I should explain a little about the statement above. The <#?> message takes a single parameter, which in this case happens to be a block. The <#?> method will send the parameter the message <#value> if the message receiver (self) is identical to the <nil> object.

Note that since blocks are first class objects they understand a variety of messages. Sending a block object one of the suite of messages for evaluation will cause all the statements the block contains to be invoked. The <#value> message is one of the messages in the “evaluation” protocol’s suite that can be used to evaluate a method or a block.

So, in this case if the <dragTargetData> is <nil> the block “[^self]” will be evaluated. If it is evaluated it will simply return from the <#dragWithin> call-frame with the result value being self; which is the original message receiver (i.e., the TEFIELD instance). Otherwise the <dragTargetData> will be assigned to the local method scoped variable <data>.

```
"Compute the offset into the text field"
offset := self offsetOfPoint:
(localPosition := Mouse localPosition).
```

Since the drag-package is interesting we proceed with normal processing. First, we will find the character offset (into our text-field) that is nearest the mouse’s local (current graphic port’s) position. We will also cache the <localPosition> value so that we can use it later.

Note, again, the SmalltalkAgents drag architecture has already guaranteed that the current canvas (graphic-port) is the window containing our TEFIELD component instance.

```
data ifFalse:
[
```

The above two lines of code are performing a test to see if the <data> value is equivalent to the <false> object, and if it is then the single block parameter will be instantiated and evaluated. I should point out here that, strictly speaking, the block will not actually be instantiated because modern Smalltalk compiler’s are pretty smart and know how to optimize or inline a great deal of the language and its constructs.

In any case, we will only enter this method once because near the end of this block-scope we will set the drag-target-data to be a list of objects, including an all important instance of block closure. Thus, any time the <#dragWithin> method is subsequently called for the current drag-target, we will be able to make use of the information we set up on this initial call.

```
line := self lineAtOffset: offset.
point := self pointAtOffset: offset.
height := self heightFromLine: line to: line.
bottom := point - (1@0).
top := (bottom - (0@height)).
```

In the above code we perform some basic pre-fighting to convert the mouse location into a given line index. We also compute the spatial characteristics of the line to enable us to accurately draw a text-caret while dragging the package around.

```
data :=
{
false.           "Not Visible"
offset.          "old offset"
top.             "top"
bottom.          "bottom"
self hiliteRegion. "The hilited text region"
[:doShow |
((data at: 1) = doShow) ifFalse:
[
"Draw in XOR mode"
activeCanvas
pushPen;
penMode: #patXor;
movePenTo: (data@3);
```


Cross-Platform Object Database Engine

neoAccess™

Easy to Use NeoAccess is a set of easy to understand C++ classes that extend the capabilities of **PowerPlant 1.0**, **MacApp 3.1** and the **THINK Class Library 2.0** on the Macintosh and **MFC 2.5**, **ObjectWindows 2.0** and **zApp 2.1** in Intel-based environments. Suddenly persistent C++ objects are first-class citizens that can be organized and accessed naturally using an API designed for minimum visible complexity. And to make things even easier, NeoAccess comes complete with **full source code**.

Full Featured NeoAccess 3.0 is now available. It includes support for threads, blobs, part lists, iterators, swizzlers, temporary objects, multiple indices on a class, a powerful relational object selection mechanism, duplicate keys, a versatile streams I/O model and incredible performance. NeoAccess has a huge storage capacity with no database administration to worry about. And databases are standard document files with your application icon on them. They are also binary-compatible across platforms.

High Performance The use of binary trees and an efficient object caching mechanism means very fast access to objects and reduced file I/O. Only objects of immediate interest need to be in memory, so your applications can be much smaller, even when accessing huge databases.

Proven Hundreds of commercial and in-house C++ developers are using NeoAccess to develop powerful CAD/CAM, multi-media, document management, data visualization, accounting, CD-ROM titles and many other kinds of applications. NeoAccess is what your organization needs to realize its potential.

Affordable NeoAccess's best feature is its price. Our development tools have always been the best value available. The NeoAccess Developer's Toolkit sells for just \$749 per developer with **no runtime licensing fees**. It includes **full source code**, numerous sample applications and 400+ pages of well-written documentation.

Native **PowerPC** support too!

neo•logic

In order to survive, you need to persist.

No Runtime Licensing Fees!
Use NeoAccess in all your development for one low price!

1450 Fourth St. • Suite 12 • Berkeley • CA • 94710 • (510) 524-5897 • AppleLink: NeoLogic

```
drawLineTo: (data@4);  
popPen.
```

```
"Update to reflect current state"  
data at: 1 put: doShow.
```

```
1  
}
```

In the above code we created an instance of <List> using the "(" and ")" dynamic list operators. The dynamic list operator will build the list by executing each statement inside the list and then collating the results from each statement.

Our list will consist of 6 elements consisting of: (1) a flag indicating if the text-drag-caret is currently visible; (2) the offset of the mouse last time the text-drag-caret location was computed; (3) the top coordinate for where to draw the text-drag-caret; (4) the bottom coordinate for where to draw the text-drag-caret; (5) the region encompassing the text-fields (hilit) selection; (6) a block closure that will both draw or erase the text-drag-caret and which will also update the flag indicating whether the text-drag-caret is visible.

```
Mouse dragTargetData: data.
```

Here we are storing the list (<data>) into the drag target reference object. By doing so, we are replacing the <false> object that was there. Remember that we can tell which of the three states that the reference object is in because we can easily discriminate between it being <nil>, <false>, or an instance of <List>.

```
1
```

Ok, by here we have done our pre-flighting and we have initialized the data structure we need for handling the steady-state <#dragWithin> operations.

```
"If the mouse is inside the hilit selection, then hide  
the caret"  
(Mouse dragInitiator == self)
```

```
and: [(data at: 5) containsPoint: localPosition]) ifTrue:  
{  
  ^(data at: 6) value: false  
}
```

In the code above we are checking to see if the receiver (our TField instance) is identical (=== test for the same object as) to the object which initiated the drag operation. If so, then we check to see if the character position that the cursor is over is inside the selection region. The purpose for this check is to catch the case where the drag was initiated from the receiver and the cursor (mouse) is currently over the text-selection which was originally dragged by the user. In this special case we want to hide the text-drag-caret and exit. We accomplish this hide and exit by evaluating the "cached" block-closure with the parameter <false>. The cached block will see the parameter as its block (scoped) temporary variable <doShow>, and will hide the caret accordingly.

```
"If it hasn't changed then just toggle/blink the vertical bar"  
(offset = (data at: 2)) ifTrue:  
{  
  "Re-draw, toggling the visibility"  
  (data at: 6) value:  
    ((Clock ticks // Gestalt DoubleTime) isEven).  
}
```

At this point we are in the middle of a keyword message for <#ifTrue:ifFalse:> where each parameter to the message is a block closure. The receiver statement will be tested for equivalence to <true> and if it matches then the first block will be evaluated, if it doesn't match then the second block will be evaluated.

The purpose of this test sequence is to enable us to discriminate the cases where the cursor has moved, from the cases where the cursor is in the same position but should be blinking on/off.

We detect the case where the cursor hasn't moved by comparing the current <offset> against the offset the last time the text-drag-caret location was computed. If the offset hasn't changed then the mouse hasn't moved; so we evaluate a drawing block and tell

it to draw the text-drag-caret based on whether the current clock ticks scaled by the Mac OS DoubleTime value is an even or an odd value. This latter test is a trick that enables us to avoid having to access extra state information. It also means that in a threaded situation it is easy to guarantee that the time period during which the caret is displayed will be essentially the same as the period during which it is not displayed.

```
"Otherwise, erase the old position and recalculate"
ifFalse:
```

```
[
```

```
"Erase the old position"
(data at: 6) value: false.
```

```
line := self lineAtOffset: offset.
point := self pointAtOffset: offset.
height := self heightFromLine: line to: line.
bottom := point - (1@0).
top := (bottom - (0@height)).
```

```
data
  at: 2 put: offset;
  at: 3 put: top;
  at: 4 put: bottom.
```

```
"Re-draw, toggling the visibility"
(data at: 6) value: true.
```

```
]
```

In this second (ifFalse:) block we are handling the case where the cursor (mouse) has moved and we need to recalculate where the caret should be displayed. The first step is to erase the old caret. Then we calculate its new location and then re-draw the caret.

Well, we are really cooking now. At this point I was about an hour into my design and was able to see all the drag tracking operations working smoothly in my text fields. I should point out that during all this activity I was inside my Smalltalk environment simply adding these operations as extensions. I never had to quit or exit and I was able to instantly see if my code failed or not. During the design of the <#dragWithin> method I had a few code errors that caused exceptions, but the dynamic environment trapped the errors and halted the erroneous threads. I was then able to debug the code and make corrections while all my applications continued to run.

I should point out that more code is actually involved in the <#dragWithin> method if we want to give text fields the ability to perform auto-scrolling during a drag. The code for that has not been presented because it would have made this article longer than necessary to illustrate the salient points of drag and drop.

So, we have two more methods that we need to design. We need a method to handle the case where an item is dropped and we need to be able to initiate a drag operation from a styled text-edit field.

We will now go over the code for handling a drag item being dropped in a styled text field. Before we do that, however, I need to mention that this method is where I spent that largest portion of my efforts because the animation and graphic rendering synchronization of the drag manager and my drag-drop animation was more tricky than I originally thought it would be.

```
TEField[i]dragItemDropped
```

```
| selEnd offset dropPoint dragData selStart result |
```

Again we declare the method temporaries (local variables) that we will need during execution scope this method.

```
"Grab the target data, if any"
```

```
(dragData := Mouse dragTargetData) ? [^self dragExit].
```

Again, we check to see if we have any interest in the drag package. We always are

MacForth Plus & Power MacForth



The Language of Innovation is now available native on the Power Macintosh.

FEATURES

- Royalty Free Turnkey Compiler
- Text Editor
- Online Glossary Tool
- System 7 Compatibility
- 68020 Assembler
- 68881/2 Co-processor Support
- High Level Graphics
- Toolbox Support
- Extensive Documentation
- Reasonable Upgrades
- Optional Tools Disks
- MacForth Plus \$199
- Power MacForth \$199



Creative Solutions, Inc.

4701 Randolph Road, Suite 12
Rockville, MD 20852
301-984-0262 or 1-800-FORTH-OK (orders)
Fax: 301-770-1675 Applelink: CSI

**Power Mac and
Macintosh
Developers:**

**FIND OUT FAST
WHAT'S GOING ON
IN MEMORY**

THE MEMORY MINE™

- See memory allocation in any open heap at a glance.
- Easily spot memory leaks.
- Flags heap corruption when it happens.
- Works with source level debugger to let you find memory problems fast.
- Stress applications on the fly with Purge, Compact, and Zap.
- Allocate memory at will for precise stress testing.
- Log heap data - easily document heap status over time.
- No need for source code: nothing inserted in code; no patches to the system.
- Works with 24-bit, 32-bit, and modern memory managers.

For Macintoshes with 68020 or better. Requires System 7.0 or later.

only \$99 US

Order now from Adianta, Inc.


Phone: (408)354-9569 • FAX: (408)354-4292

AppleLink: ADIANTA • AOL: Adianta • Internet: adianta@aol.com

For VISA, MC, or American Express orders by mail, fax, or Applelink, please include name, address, card number, expiration date, and phone number or email address.

Also available through the MacTech Mail Order Store.

for more information contact

 **Adianta, Inc.** • 2 N. Santa Cruz Ave. #201 • Los Gatos, CA 95030

Databases in the Finder?

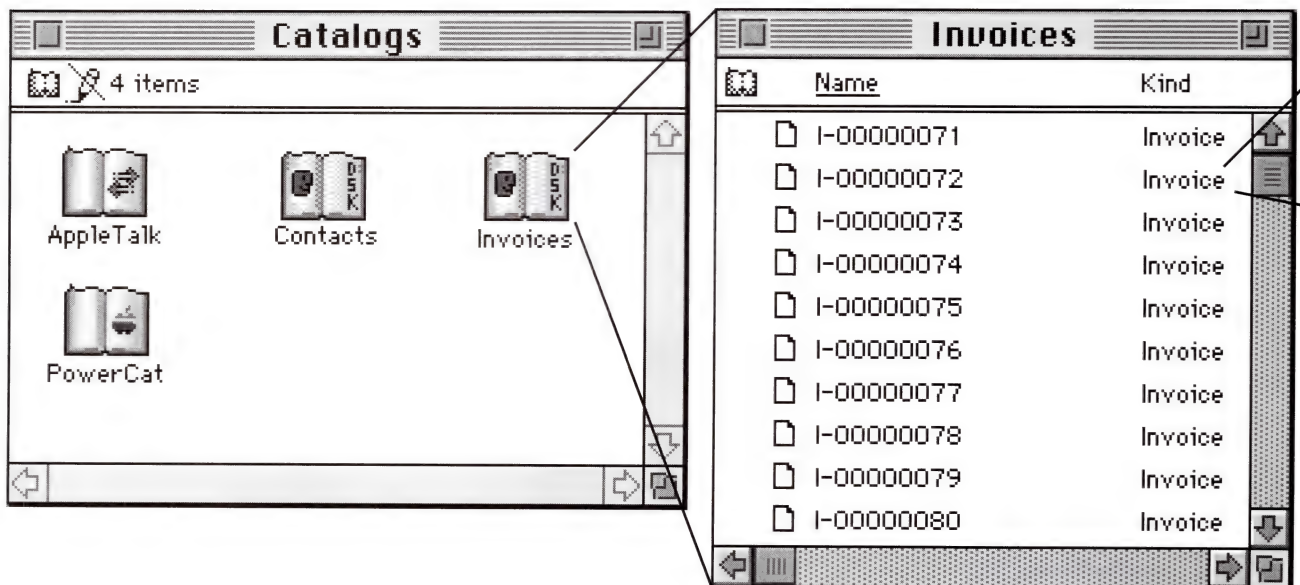
Unfortunately, most users must interact with relational databases at the SQL level or behind a front-end that can take months to develop.

```
select invoice_num,invoice_date,cust_id,cust_name from invoices,customers where invoice_id = 112;printall;
```

Why not view and edit your data at the Finder level? Announcing **Cataloger™**.



Cataloger brings a whole new way of working with databases to the Macintosh. The Finder™ has always provided a view of our desktop files, why not database records too?



Apple's **Open Collaborative Environment** places a catalog icon at the Finder. Open the icon up and you'll see a window like the one above. The AppleTalk catalog shows nodes on the network. The PowerCat catalog shows users and groups in a PowerShare Server.



The Contact and Invoices catalogs (for example) can show information from **4D Server**, **DAL/DAM**, **Oracle** and **Sybase** databases! If your needs are more customized, you can write AppleScript handlers to provide the contents of your catalog.



The Catalog Manager (an AOCCE API) allows for **Catalog Service Access Modules** (CSAMs) to be created that interact with external sources.



The Finder and other applications make requests through the Catalog Manager and Cataloger does the rest. There are no limits to the amount of data brought back.





Templates provide the ability to view and edit individual records. AOCÉ Templates are designed with **Template Constructor™**. This powerful application let's you easily design forms to view data coming from several sources, control behavior with C and AppleScript, and view row/column data with **TableIt!™** - a powerful data display tool.

I-00000072

Invoice #: I-00000072 **Invoice Date:** 9/1/94

Company: Tall Timbers Marina **Terms:** Net 30 Days ▼

Qty	Part #	Description	Cost Each	Line Total
1	2143	Propeller	250	250
2	3853	Drain Plugs	12.90	25.80
1	4905	Main Light assembly	150	150
8	9384	Spark Plugs	4.00	32.00
1	9999	Labor	600	600

Notes: Prepared boat for summer season.
Dewinterized, tuned up and replaced main propeller. Light assembly replaced after testing front lights failed.

Subtotal: 1057.80
Tax: 52.89
Total: 1110.69



Access information and catalog definitions are stored in and protected by the AOCÉ **Keychain**. Your users can have one username and password for a variety of data sources!



Need more horsepower? **Database Scripting Kit™** provides AppleScript access to all of the connection information and data sources. Write AppleScript routines that send queries and parse results from within your own application.

Cataloger/Template Constructor includes:

Database Scripting Kit™ with connectors for 4D Server, DAL/DAM, Oracle, Sybase and others.

DSK Cataloger™ CSAM

Template Constructor™ with TableIt!™

Several example catalogs/templates and over 100 example AppleScripts.

Complete scriptability, native for Power Macintosh. Drag Manager and Thread Manager support.

How to contact us:

**Graphical Business
Interfaces, Inc.**

Voice: 800-424-7714 or 219-253-8623

Fax: 219-253-7158

E-Mail: steve@gbi.com

GBI. Bringing your data into view.™

notified of a drop, even if we have no interest in the package itself. The drag manager can only determine if we had an interest in the package itself when we request data from the package during the processing of the <#dragItemDropped> method.

Knowing this information is useful because it allows the drag manager to automatically inform the drag-initiator what the disposition of the drag package was. I should also point out that this functionality is outside the scope of the Apple Drag and Drop Manager's architecture.

```
dropPoint := Mouse localPosition.
offset := dragData at: 2.
```

In the code above we obtain the actual drop-point for the package and then we recover the last known mouse location from our drag-within <data> list. For a variety of subtle reasons these may not be the same value and we will need to have both to properly clean up the visual state of the desktop.

```
self dragExit.
```

We now issue a <#dragExit> message which will put us in a clean state for the <#dragItemDropped> operation. The <#dragExit> operation will be called twice because of this usage. The first time is our call above, the second time will be when we return to the drag manager after this <#dragItemDropped> method completes. We know this is safe because both the <Mouse> drag methods and the <TEField> methods that we wrote don't perform unnecessary actions (like unhilting the drag-region if it was previously hilted). This kind of added flexibility provided by using safety checks was part of the basic design architecture of the SmalltalkAgents drag mechanism.

```
"If we were the drag initiator, and the drop was inside our
hilted text, then do nothing."
(Mouse dragInitiator == self
 and: [(Mouse dragInitiatorData@1) & 0x44] not
 and: [Keyboard AltKey not]]) ifTrue:
```

The code is testing that three conditions are true: (1) That the drag-initiator is the same as the drop-target; (2) that the drag was not initiated with using the ALT/OPTION metakey [which indicates a copy]; (3) that the ALT/OPTION metakey is currently not pressed. Clearly we are looking at code that was modified based on knowledge about the way the drag-initiation was going to function. There is also a certain redundancy here with regard to the implementation of option-copy semantics which may be further refined in future versions of this architecture.

The 0x44 is a portable mask we can use for testing whether the left or right option keys are pressed on the keyboard. The SmalltalkAgents virtual machine defines a universal keyboard (i.e., a portable key-code system) to ensure that key-code operations can be written independently from the type of host-operating system or host-hardware.

If the above three conditions are true then we are processing the complicated case of dragging within the same component (i.e., container) and the operation is therefore a move not a copy. The move operation is tricky because we need to preflight some calculations so that our drag animation from the selection's current location to its new location will be uniformly smooth.

```
((dragData@5) containsPoint: dropPoint) ifTrue:
{
  ^self
}
```

In the above code we are testing to see if the drop point is located over the source-selection. If it is then no action is required because we don't move text on top of itself. Therefore in this situation we simply exit because we are done.

```
selEnd := self selectionEnd.
selStart := self selectionStart.

(selEnd selStart) ifTrue:
{
  | line height point |
```

Now we begin the calculation to see which of two possible drag animation situations exists. The above code is doing a pre-flight to compute the selection ranges. Strictly speaking the $selEnd \geq selStart$ test is not needed because there must be a selection or we (presumably) would never have initiated the drag (however, it is here for safety).

The `| line height point |` block temporary variables are declared here and are locally scoped to only be defined within the block itself.

```
line := self lineAtOffset: offset.
height := self heightFromLine: line to: line.
```

```
point := self pointAtOffset: offset.
```

```
"If on the same line then remove the width from
the offset because it will be cut anyway."
((offset selStart)
 and: [(self lineAtOffset: selStart) = line])
  ifTrue:
{
  point := point - ((dragData@5) bounds
    width + 1)@height.
}
ifFalse:
{
  point := point - (1@height).
}
```

In the above code we have preflighted the calculation of the line, its height, and the baseline of the area where the text-drag-caret was displayed. We use the caret location because we can be sure that it is located precisely over a character even if the drop-point was not. This situation can occur when the drop point is below the last line or to the right of the last character in a given line.

If the character drop location is to the right of the selection and it is on the same line then we need to adjust the animation destination because the target insertion point will move by the number of characters that we are "cutting" from the display.

What we are trying to do here is to define a starting and ending path for our drag animation. The starting point is the current selection region (in `data@5`). The ending point has to be calculated based on whether the insertion point will shift after the cut operation.

We subtract the height of the destination line so that the drag path will be based on a top-left base difference between the current selection origin and its final after the clear and insert has been performed.

```
"Animate the move operation"
(dragData@5)
  zoomBy: (point - ((dragData@5) bounds origin))
  mode: 1
  steps: 12
  rate: 12.
```

Now we use the Smalltalk drag animation method that will animate an arbitrary region along a linear path. The path begins at the receiver's origin and moves for some delta distance based on the "zoomBy:" parameter. The receiver to the above animation message can be any kind of <GraphicPrimitive> object, so our selection-region is appropriate here.

Apple's drag manager would be substituted here for drag animation but since we had our own with a bit more tuning flexibility, I used the SmalltalkAgent's one instead. The Smalltalk animation method has finer control of steps, and of the acceleration in terms of both the step scaling and the step rate.

```
"Clear the existing selection"
self doClear.

"Adjust for the portion we remove"
(selEnd offset) ifTrue:
{
  offset := offset - (selEnd - selStart).
}
}
```

```
"Select the insertion point"
self selectFrom: offset+1 to: offset.
```

The code above has cleared the current selection thus removing it from the text field. Then we adjust the offset for the insertion point to account for the cleared text. Finally we set the new insertion point in preparation for our insert (i.e., preparatory work before we can issue a <#nextPutAll:> message) operation.

```
} ifFalse:
{
  "Revise the insertion point"
  self selectFrom: offset+1 to: offset.
}
```

The `ifFalse:` block parameter is the case where it was not a self move, but rather a copy operation. In this case we want to leave any current selection intact and thus we just set the insertion point to where the user indicated the character drop point should be.

```
"Insert the data"
```


The professional's database engine

CXBase Pro

for
Macintosh and Power Macintosh

Based on C-Index Pro, the database engine used by **Apple AOS** for their **eWorld** content publishing software, *CXBase Pro* has all the features and power of C-Index Pro, plus additional new capabilities. *CXBase Pro* includes a flexible select module, access to both logical and absolute record numbers, and more.

CXBase Pro is designed as a "pure" database engine. The philosophy behind *CXBase Pro* is to provide a simple, consistent, and powerful API that lets you work in whatever way you choose. *CXBase Pro* is delivered with full source code, giving you maximum flexibility and security. The *CXBase Pro* library provides both multi-keyed data record handling, and flexible BLOB storage where you define the data format. And *CXBase Pro* has been engineered from the ground up for maximum performance.

■ New! Powerful and flexible select module

CXBase Pro lets you create multiple kinds of selections, and provides a callback mechanism that allows you to both display the progress, and capture the data as it is selected, giving you full control over the operation of the select routine.

■ New! Access to logical and absolute record numbers

You can access data records not only by key, but also by logical and absolute record numbers. This gives you maximum flexibility as to how you retrieve and display your data.

Additional standard features:

■ Unrivalled performance - runs native on the PowerPC!

■ Royalty-free license

■ Complete ANSI C source code provided

■ Data files and code 100% cross-platform

Full-featured \$49 demo available!

CXBase Pro also comes in a demo version. You get the complete manual, and a full-featured library limited only by file size.

For full prices, to order, or to receive more information contact:

TSE International | Taandwarsstr. 51 | 1013 BV Amsterdam, Holland | tel: 31 20 638-6507 | fax: 31 20 620-4933 | AppleLink: TSE.INT

```
Mouse dragItemsDo:  
[:index :typeList | mapType |
```

The block above takes up to two parameters and declares one block temporary variable to hold the map type. What this method is going to do is look at each of the objects in the drag-package and extract out all the objects that we can coerce to some form of <String>. This block is evaluated, by the <Mouse> object, once for each of the different clipping groups in the drag package.

A clipping group is a list of "types" (i.e., formats) that are available for a given promised object. This mechanism is the same technique that is used for Macintosh clipboard entries where there might be a pixel-map, a picture, and an icon which form a clipping group that represents a Finder copy of a desktop icon.

```
mapType := nil.  
(typeList includes: Text) ifTrue:  
[  
    mapType := Text  
]  
ifFalse:  
[  
    (typeList includes: String) ifTrue:  
    [mapType := String].  
]
```

In the above code we examine the <typeList> for the clipping group and determine if there is an object of interest in the group and if so we record its type so that we can extract it. The SmalltalkAgents environment handles data type coercions so if we asked for a <String> when <Text> was available the framework perform the appropriate coercion (this is similar to the AppleEvent type-coercion system).

```
mapType ifTrue:  
[  
    result ifNil:  
    [  
        result := Mouse
```

```
extractDragItem: index  
asInstanceOf: mapType.  
] ifNotNil:  
[  
    result := result, (Mouse  
        extractDragItem: index  
        asInstanceOf: mapType).  
]
```

In the above code block we append the extracted text or string data from the current clipping group onto the result. The Smalltalk class for <Text> preserves style information when concatenating (via the <#> message) so we don't have to pay attention to the complexities of Macintosh styl/text resource type concatenation.

```
].  
  
result ? [^self].
```

If there was no data extracted then <result> will be <nil> and which means that we have no more work to do so we exit returning the receiver (self) as the method's result. In Smalltalk a method always returns an object which enables all message operations to be treated uniformly. The default return value, if there is no explicit return value, is the message's receiver <self>.

```
self  
nextPutAll: result;  
selectFrom: offset to: offset+ result size.
```

Now we insert the drag-package data we extracted and then we select the inserted data and we are done!

Well, we are really on the way now! At this point I had spent approximately 3 hours implementing drag and drop and was ready for the final operation which was initiating a drag

operation. Prior to writing my own initiator I had been dragging from the sample drag tools that Apple provided with the Drag and Drop developer kit from APDA.

In fact, we are almost done, although I didn't realize it at the time because I thought it would be a lot harder to build the drag initiation method than it turned out to be. The `<#handleDragEvent:>` method involved a minor modification to my existing `<TEField>` button-press method for so that I could detect when a single-click had occurred over a selected range of text. This test is the basis I used for determining whether the user intended to drag a text chunk around.

I have deleted portions of the source code from the method that follows to help you (the reader) maintain your focus on the portions of the code that were relevant to the styled text field's drag and drop implementation.

```

... Portions Deleted ...
TEField[i]buttonPress: event

(Switch new)
case: 1 do:
[
(self handleDragEvent: event) ifFalse:
[
<<TEClick(localWhere:long;
(event ShiftKeyOnly):Boolean; self:Handle)>>.
]
]

```

What we have done here is to test for the case where a single-click occurred. Then we pass the event to our `#handleDragEvent:` method and see if it handled the button-press operation. If it did not then we process the button-press as if drag and drop did not exist.

```

}

... Portions Deleted ...

on: event clickCount.

self
updateScrollers;
"scrollSelectionIntoView;"
privateSetCursor.

canvas popPen.
thread popCanvas.

```

```

TEField[i]handleDragEvent: event

"Implement the drag drop defaults"
((self->selStart) (self->selEnd)) ifTrue:
[
| dragRegion |

```

In the code above we are accessing the structured storage of the `TERecord` and testing to see whether the selection range includes any characters. If it does then we enter this block and begin testing to see if the button press event location occurred over top of the current text selection.

```

((dragRegion := self hiliteRegion)
containsPoint: (event localWhere)) ifTrue:
[

```

If the hilited text selection contains the event's local-where point and mouse button-1 is pressed then we fix up the starting location for the drag operation to ensure that any event processing latency will be corrected as we build our drag image.

```

[(Mouse isButtonDown) and:
[(Mouse localWhere maxDelta: event localWhere)
2]] whileTrue.

```

Now if `<Mouse>` button 1 is still down, then we construct a drag package; an image to drag around the screen; configure drag initiation data. We also configure the drag flags such that the drag will block our initial hiliting and disable auto-centering of our drag image.

The `<drag:>` parameter is a clipping group that we are offering for dragging. There are other variations of the drag initiation command that the `<Mouse>` supports that allow multiple items to be dragged.

The `<withImage:>` parameter can be any arbitrary graphic object that conforms to some basic rendering protocols (i.e., responds to a predefined set of messages). In this case we use the current selection region and create an outline for the user to drag around.

The initiator data is private for the use of the drag initiator and can be anything. We currently use it to hold the meta-key state at the time of drag initiation and also to hold the original drag region for text movement (as opposed to copy) animation.

```

Mouse isButtonDown ifTrue:
[
Mouse
drag: (self selectedContents)
withImage: (dragRegion copy differenceWith:
(dragRegion copy insetBy: 1))
startingFrom: event localWhere
initiator: self
initiatorData:
{Keyboard metakeys. dragRegion}

"Don't block initial hilite and don't
auto-center"
flags: 0x03.
^true

```

Since we initiated a drag operation we signal that we have handled the button press event by returning `<true>` as our method result.

```

}
}
^false

```

The `SmalltalkAgents` flags parameter allows us a finer grain of control than that which is provided by Apple's Drag and Drop Manager. Specifically the flags enable us to control the drag tracking pre-flight operations, and also let us control where the dragging can occur. Currently the flags allow us to restrict dragging to within the initiators window, to the module that the window belongs to, to the Smalltalk environment that the window belongs to, or to the entire Macintosh Application space of the desktop.

If the dragging is not to the entire Macintosh Application space, or Apple's Drag and Drop Manager is not available, then dragging will be completely handled in Smalltalk. The `SmalltalkAgents` Drag and Drop Architecture functions in a portable fashion that is independent of the presence of the Apple Drag and Drop Manager.

Mouse Drag Flag Definitions

```

0x0001 .. Do not initially hilite the drag-source
component
0x0002 .. Do not auto-center the drag-image
0x0010 .. Restrict dragging to the source Environment
0x0020 .. Restrict dragging to the source Module
0x0040 .. Restrict dragging to the source Window"

```

Well, that's it. The code is all that was needed to add drag and drop with animation of dragging and the drop-moving of text selections. I hope this article helped to shed some light on the subject of how such things were done in a language like Smalltalk. The best part for me in doing all this was that it was

really fun watching it come alive as I worked on a running application environment. Even when I had bugs, they were caught and I could continue my work in an uninterrupted flow. The dynamic, interactive nature of the development process really allowed me to fine tune the user interface behavior to get exactly the effects I wanted.

CLOSING

Overall, the adding of drag and drop to the <TEField> class took me about 4 hours from design concept to end result. Prior to doing the design and implementation, I spent about 2 days looking at other drag and drop behavior and implementations in various applications on different operating systems. After I was done with the implementation I let it sit and "breathe" for a few days and then I came back and spent some 2-3 hours interactively experimenting with variations on the animation and cursor hilite-tracking aesthetics.

Once we had drag and drop available in-house, we found ourselves using it all the time to create clippings. For example, we use it as an extended clipboard for code editing and saving operations – specifically we create useful "snippets" of code or tools that we can then compile & execute (evaluate) any time during the development process.

This has also led to a whole avenue of other unforeseen possibilities such as clipping scrapbooks, and the use of clippings as a documentation tool for our Platform Independent Portable Object (PIPO's) packages. A PIPO is an arbitrary collection of objects that can include executable code. SmalltalkAgents PIPO mechanisms provide a real-time persistent storage, streaming, and object linking mechanism.

In the latter case, we might have a resource of type PIPO in the clipping file, and we might also have a styled text resource. The Finder would ignore the PIPO resource and would just display the styled text thus enabling the text to be used as a means of displaying a description of the other clipping contents (i.e., the PIPO).

The Finder is quite flexible in the way it handles clippings because it allows the clipping file-creator to be any type; it identifies a clipping by the file-type of clip. With a little ingenuity, an application can use the custom-icon feature of System 7 to set the clipping file's icon to use their own stylized version to enable differentiation.

Adding a version 2 resource in the clipping file might also be a desirable behavior so that the clipping creator can be properly identified from a Finder "Get Info" window.

FEEDBACK

The author can be reached at "David_Simmons@qks.com". The electronic mail address "info@qks.com" is available for any product questions you might have. For more general information on SmalltalkAgents you can access:

Relational Database Classes

RC/21

- **Multi-Platform**
- **Client/Server or Embedded**
- **Built-in Relational Engine**
- **FAST Development**
- **FAST Execution**
- **Source Included**

**Vermont
Database
Corporation**

1-800-822-4437

Vermont Database Corporation
400 Upper Hollow Hill Road
Stowe, VT 05672-4518
USA

Voice: 1-802-253-4437
Fax: 1-802-253-4146
vtdatabase@vermont.com
CIS: 70334,3705

WorldWideWeb:

The QKS World Wide Web site "<http://www.qks.com/>".

INTERNET:

Anonymous ftp via "ftp.qks.com".

Compuserve:

"Go Smalltalk"

or "Go MacDev [Dynamic Languages Section]"

or "Go MacDev [Object Oriented Section]"

There are also a number of electronic discussion groups that talk about Smalltalk, and some that are dedicated to SmalltalkAgents. QKS provides its own "STA-Forum@QKS.COM" e-mail forum that anyone is welcome to join (for details send mail to either "listserv@qks.com" or "postmaster@qks.com").

STANDARD DISCLAIMERS

SmalltalkAgents is a registered trademark of Quasar Knowledge Systems, Inc. All other brand or product names mentioned are trademarks or registered trademarks of their respective holders.



This monthly column, written by Symantec's Technical Support Engineers, aims to provide you with information on Symantec products. Each month we cover either a specific application of tools or a "Q&A" list.

Q. *Can I use the Think Class Library in a CODE Resource?*

A. In case you haven't noticed, CODE Resources are finicky. The TCL uses virtual functions, and the virtual table uses global data space. Using global data space is upsetting to the discriminating palette of CODE Resources which use A4. You can, however, use your own classes that do not use virtual functions. If you decide to do this, you must turn on the Multi-Segment option.

Q. *When I run Profiler, why do I get an internal error in PC.H?*

A. This is a bug. When you run Profiler, turn off "Use function calls for inlines", under the options for the C++ compiler, in the compiler options window.

Q. *Why is a CArrowPopupPane arrow not drawn in the center of the pane?*

A. The horizontal setting is incorrect in the Draw() routine. The trick is to set the field `signPt.h` to 0;

```
void CArrowPopupPane::Draw(Rect *area)
{
    Point signPt;
    signPt.h = 4;    // set this to 0;
    signPt.v = 7;
    DrawSIGN(TCL_SIGN, POPUP_SIGN, signPt);
    CPopupPane::Draw(area);
}
```

Q. *Why do derived classes of ifstream cause a bus error when they read in a stream?*

A. Any class indirectly derived from the virtual base class `ios` needs to explicitly call the `ios` constructor to initialize the buffer into which the data stream is read. For example:

```
class myifstream : public ifstream {
myifstream(char * s) : ios(&buffer), ifstream(s){}
```

Q. *When I create a text file using ofstream, why can't Mac word processors open the file?*

A. In order for a word processor to recognize the text file, the file must have a recognizable file type and creator. First, you will need to include `stdio.h`. Second, you will need to initialize the globals `_ftype` = 'TEXT' and `_fcreator` = 'ttxt' before creating the text file. For example:

```
#include <stdio.h>
#include <fstream.h>

void main(void) {
    FILE *fp;
    _ftype = 'TEXT';
    _fcreator = 'ttxt';

    ofstream textfile ("foo.txt",ios::translated);
    textfile << "This is a test" << endl;
    textfile << "This is line 2\nThis is line 3\n";
    textfile.close();
    return 0;
} //end main
```

Q. *Why do I get incorrect values for floats when I have the 68881 options on?*

A. When you set the 68881 compiler options for your project:

- open all the libraries
- the libraries and the project must have the same options set
- bring the project up to date

Don't forget the `unix++` library that is in the IOSTreams library.

Q. *I am using the example in the manual for using the #pragma parameter as part of the process of locking a CODE Resource. Why do I get the error that a prototype is required?*

A. The first statement generates an error. The second statement should resolve this error.

```
#pragma parameter __D0 RecoverResPtr()
Handle RecoverResPtr() = 0xA128;    //generates error

#pragma parameter __D0 RecoverResPtr(void)
Handle RecoverResPtr(void) = 0xA128;    //resolves problem
```

Q. *When I am using the debugger to debug my program when the project, project files, and debugger are not all on the same machine, why do I get a message in the debugger, "Out of memory"?*

A. The debugger was not originally designed to work on remote files. The source files will have to be on the same machine as the project file and Debugger.

Q. *I am using MacApp with SC++ for MPW. The MacApp documentation states that I need to rebuild the Symantec Libraries with model -far. How do I do this?*

A. You should be able to compile most MacApp projects without modifying the SC++ libraries. However, if you really want to rebuild the libraries, follow these steps:

C/C++ without Object Master



Introducing a powerful new way of looking at code development.

Open your eyes to Object Master™, the most innovative programming tool available on the market today. With its powerful editors and intuitive windows, Object Master gives you the unsurpassed freedom to develop code quickly and accurately.

A Real Eye Opener

Use Object Master's unlimited number of browser windows to access code components and display their definitions, ready for editing. All changes you make in the browser windows are automatically

displayed throughout the environment, consistently ensuring current and accurate code. With the browser windows, you can also view a class list displaying the

hierarchical relationships between classes, and use pre-made templates to create classes and methods.

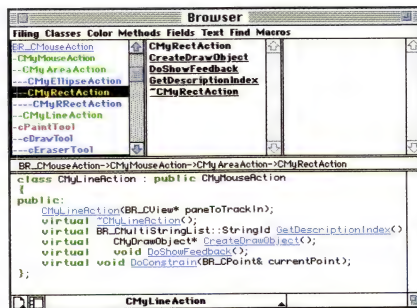
Picture Perfect Code

While the browser windows let you edit specific pieces of code, Object Master's File editor gives you a full-file view of your code, allowing you to quickly perform universal edits.

To help you identify important pieces of code easily, Object Master color codes and formats language elements. With a single keystroke, Object Master will look up parameters for methods or functions contained in the project and paste them directly into your source code.

Improved Insight

Don't worry about the physical location of your code—Object Master parses all files and maintains a data dictionary of project components. The dynamic environment updates your entire project automatically as you edit without compilation!



True browser windows let you see code like never before.

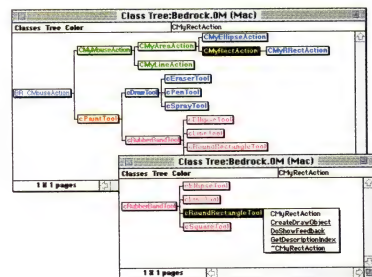
Object Master provides you with all the editing and navigational capabilities you require to write code productively. For example, the Class Tree window graphically displays your project's class hierarchy and allows you to expand and collapse "branches" and open multiple windows displaying specific code components.

Power Macintosh Support

ACI offers two versions of this outstanding programming tool: Object Master and Object Master UNIVERSAL. Both run on the traditional 68K-based Macintosh computer and the new Power Macintosh, taking full advantage of the features of each platform.

Object Master supports C and C++ and works seamlessly with Symantec's THINK Project Manager and Metrowerk's CodeWarrior. Separate versions are available for the 68K-based Macintosh and the new Power Macintosh computers.

Object Master UNIVERSAL supports C, C++, Pascal, Modula-2, and all major compilation systems. It can be installed on both the 68K-based Macintosh and the new Power Macintosh.



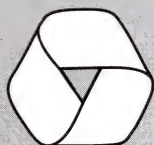
"...Object Master pays for itself in a week, even at suggested retail price."—Macworld Magazine

See Object Master for Yourself

Call (800) 384-0010, and we'll send you a free demo disk of Object Master—because seeing really is believing.



ACI US Inc. 20883 Stevens Creek Blvd., Cupertino, CA 95014
Tel. 1 408 252 4444. Fax 1 408 252 0831. AppleLink D4444
©1994 ACI US, Inc. All product or service names mentioned herein are trademarks of their respective owners. Quote reprinted courtesy of Macworld Communications, 501 Second Street, San Francisco, CA. 94107



LS Object Pascal



Powerful ■ Faster ■ Native
shipping on CD-ROM for PowerMac
now \$399 was ~~\$695~~

1-800-252-6479

Language Systems Corp. • 100 Carpenter Dr. • Sterling, VA 20164 • (703) 478-0181 • Fax: (703) 689-9593 • AppleLink: LANGSYS

Apprentice

Mac Source Code CD-ROM

Over 450 megabytes of up-to-date Mac-only source code in C, C++, and Pascal. Libraries, MPW tools, languages, utilities, information, demos, and much more! \$35 including shipping. Add \$5 for shipping outside of the U.S. and Canada. VISA, MC, AMEX, and Discover gladly accepted.



Celestin Company, 1152 Hastings Ave, Port Townsend, WA 98368
800 835 5514 • 206 385 3767 • 206 385 3586 fax
Internet: celestin@olympus.net • CompuServe: 71630,650

DEVELOPER



UNIVERSITY

Power Up with PowerPC !

Apple's Developer University has the right courses to jump start your PowerPC development efforts:

- Programmer's Introduction to RISC and PowerPC self-paced training
- PowerPC BootCamp Class

And, with over 19 courses, we offer the fastest way to get up to speed on the Macintosh, whether you're just getting started or seeking to understand Apple's newest development technologies.

For more information, contact the Apple Developer University Registrar by telephone at (408) 974-4897 or fax (408) 974-0544.

Developer University, Apple Computer, Inc. 1 Infinite Loop, MS 305-1TU, Cupertino, CA 95014

CMaster 2.0 installs inside of **THINK C 5/6/7** and **Symantec C++ For Macintosh**, and enhances the environment. No new editor to learn—CMaster builds on what Symantec has already provided. See for yourself why programmers from around the world use CMaster, and why MacUser gave CMaster 1 a **5-Mouse** rating.

CMaster 2.0 incorporates hundreds of features asked for by active programmers like yourself. These ads each highlight one special aspect CMaster 2.0 brings to the Symantec environment. EMail, FAX, or call for more information or a Demo Disk.

CMaster[®]

The Fast, Efficient Way to Edit THINK[™] Code

Rick Johnston, Eartown Movies:

I ordered the upgrade the second I saw it—I don't think I could function without CMaster!

→ Speed

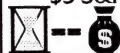
Faster programming is CMaster's primary feature: for example, writing a new function. Use the Marker and GoBack menus to roam in the current file or project grabbing code snippets, storing them in clipboard stacks. Enter the function outline, then paste back the snippets as needed. Use expandable Glossaries and Macros to speed new code development. Press a key and move to the function start or the end of the automatics, and enter some (or use the "Enter Automatics" feature.) Highlight questionable variables, and CMaster searches for usage—if unused, CMaster intelligently highlights the declaration for quick deletion. Press ENTER to *snapback* to the previous edit location to finish up. Use CMaster to produce a formatted prototype, press again, and the matching ".h" file comes foremost, or CMaster finds and opens it for you. Voilà tout!

CMaster PACKAGE ONLY \$129.95

30 DAY MONEY-BACK GUARANTEE

Jersey Scientific, Inc. 291 Springfield Av Suite 201, Berkeley Heights NJ 07922 TEL: 212.736.0406 FAX: 908.464.3458
APPLELINK: JERSCI • CIS: 70400,3361 • MASTERCARD, VISA, CHECKS, AND CORPORATE POs ACCEPTED.

\$5 S&H US Post (\$10 UPS BLUE, \$19 Intl Express) • E-mail orders accepted—send VS/MC number, exp. date, and ship address
Free Demo Disk: Available on the Internet, CSI, and AppleLink. Or FAX, Email, or Call and ask for it.



THINK is a trademark of the Symantec Corporation. CMaster is a registered trademark of Jersey Scientific, Inc.

- Go into the folder :MPW:Libraries:SCLibraries:SCSrcLibCpp and change SCLibCpp.make from: COptions= -r -O all -sym on AOptions= -case on to: COptions= -r -O all -sym on -model far AOptions= -case on -model far
- Remove the precompiled header file IOSMacDefs in :MPW:Libraries:SCLibraries:SCSrcLibCpp:IOSMacDefs
- If they do not exist, create the folders SCppObj and SCObjpp881 in :YOURHD:MPW:Libraries:SCLibraries:SCSrcLibCpp
- Do a build of SCLibCpp. Repeat the process with SCLibCpp881.make and do a build of SCLibCpp881.

Q. When I create a TCL application, is there any particular way that I need to segment my project?

A. You can segment your application in any fashion, but certain libraries must not be purged. These files and libraries must be in a resident segment:

CApplication.cp	TCLUtilities.cp	TCLpstring.cp
Exceptions.cp	CPlusLib	MacTraps2
LongCoordinates.cp	MacTraps	

As delivered, the VA project model and the demos have MacTraps and MacTraps2 in purgeable segments. This is because these projects don't contain any code that would ever unload these segments. If your code does any loading or unloading of segments, make sure that you change the MacTraps segment so that it is not purgeable.

SPECIAL THANKS TO

Craig Conner, Colen Garoutte-Carson, Rick Hartmann, Steve Howard, Celso Barriga, Kevin Irlen, Yuen Li, and Scott Shurr.



All TEXT is not created equal.

Is your application taking *cut & paste* a little too literally when it creates text? Are you tired of living in a monotype world?

PAIGE[™], our text & page layout programming library, is the ultimate cross-platform solution.

PAIGE provides the most sophisticated features/functions in the business. These include:

- **Stylized Text**
- **Shapes & Containers**
- **Text Wrapping**
- **Embedded Objects**
- **Hypertext Links**
- **Virtual Memory**
- **Style Sheet Support**
- **Multi-Level "Undo"**
- **Royalty Free**

So why should you join the **PAIGE** revolution? TIME. Most programmers don't have time to reinvent the wheel.

PAIGE was designed using no global variables and machine specific code has been isolated into two small source files. This strategy allows you to move your application to other operating systems or platforms by changing only platform specific code while maintaining full data and application compatibility.

Join the hundreds of major software publishers using **PAIGE** as their total text solution. For complete technical/pricing summary contact **DataPak Software** at 800-327-6703 or 206-573-9155.

Macintosh • Power Macintosh • Windows



dtF The Relational Database System

What is dtF...

dtF is a true relational database management system for C/C++ application development. dtF features SQL, full transaction control, error recovery and client-server architecture.

dtF is royalty free...

Create and distribute as many applications as you like, royalty free!

dtF is fast...

When it comes to performance dtF is in a class all its own. dtF utilizes a proprietary query optimization and caching scheme to obtain unparalleled performance.

dtF is efficient...

dtF was developed by Macintosh developers for Macintosh developers. Applications developed with dtF will be compact in both single and multi user form. Perfect RDBMS for Powerbook applications.

dtF is SQL...

dtF supports an efficient subset of ISO standard SQL that is optimized for speed and ease of use.

dtF is safe...

Full transaction control and error recovery guarantee maximum data protection even after sudden system crashes. dtF databases are compressed and encrypted to protect against all unauthorized access, even disk editors.

dtF is easy...

Integrated data dictionary, security, automatic index selection, query optimization, deadlock detection and error recovery allow you concentrate on your application.

dtF is true client server...

dtF client server architecture insures efficient use of network bandwidth and optimum data security. dtF will not bog down your network like systems that use file sharing.

dtF single user...

Client server applications will not be stranded on the LAN. Relink your application with dtF single user and you will have a no compromises high performance stand alone.

dtF is for you!

For maximum performance, realistic licensing policy and reasonable pricing you can not beat dtF. No static preprocessors, only a dynamic, fully featured SQL. The C/C++ API is identical and fully portable over all supported platforms in both single and multi-user environments. dtF is ideal for use with TCL and MacApp.

dtF Platforms and Compilers...

Available for Macintosh System 7.x and native Power PC. dtF Server requires a 68020 or better. MPW C/C++ and Symantec C/C++ 5.x and higher are both supported.

dtF Evaluation	\$129
dtF Macintosh SDK	\$695
dtF LAN Macintosh SDK*	\$1595
dtF Server	\$1295

dtF Americas, Inc.
12545 Olive Blvd, Suite 130
St. Louis, MO 63141
800.DTF.1790 Voice
314.530.1697 Fax
AppleLink: DTF.AMERICA



dtF is also available for DOS, Windows, OS/2 and several flavors of UNIX.

By Mike Scanlin, Mountain View, CA



HUFFMAN DECODING

Being able to decode a compressed bit stream quickly is important in many applications. This month you'll get a chance to decode one of the most commonly used compression formats around. Huffman codes are variable length bit strings that represent some other bit string. I'm not going to explain the algorithm here (see any decent book on algorithms for that) but I will explain the format of the symbol table you'll be given and show you how to decode using it (which is all you need to know to do this Challenge).

The symbol table you are passed consists of an array of elements that look like this:

```
typedef struct SymElem {
    unsigned short  symLength;
    unsigned short  sym;
    unsigned short  value;
} SymElem, *SymElemPtr;
```

where sym is the compressed bit pattern, symLength is the number of bits in sym (from 1 to 16, starting from the least significant bit of sym) and value is the uncompressed output value (16 bits). The symbol table will be sorted smallest to largest first by length and then, within each length, by sym. For example, if you had a table with two SymElems like this:

```
sym = 3; symLength = 2; value = 0xAAAA
sym = 1; symLength = 3; value = 0xB BBB
```

and then you were given this compressed bit stream to decode 1100111001001, then the output would be 0xAAAA 0xB BBB 0xAAAA 0xB BBB because the first two 1 bits are the 2-bit symbol '11' (i.e. 3), and the next 3 bits are the 3-bit symbol '001', and so on.

You will have a chance to create a lookup table in an un-timed init routine but, the amount of memory you can use is variable, from 8K to 256K. Your init routine cannot allocate more than maxMemoryUsage bytes or it will be disqualified

(this includes 'static' and global data):

```
void *HuffmanDecodeInit(theSymTable, numSymElems, maxMemoryUsage);
SymElemPtr    theSymTable;
unsigned short numSymElems;
unsigned long  maxMemoryUsage;
```

The return value from this init routine will be passed to the actual decode routine (as the privateHuffDataPtr parameter). The decode routine (which is timed) will be called with different sets of compressed data that use the same symbol table:

```
unsigned long HuffmanDecode(theSymTable, numSymElems, bitsPtr,
                           numBits, outputPtr, privateHuffDataPtr)
SymElemPtr    theSymTable;
unsigned short numSymElems;
char          *bitsPtr;
unsigned long  numBits;
unsigned short *outputPtr;
void          *privateHuffDataPtr;
```

You can assume that outputPtr points to a buffer large enough to hold all of the uncompressed data. The return value from HuffmanDecode is the actual number of bytes that were stored in that buffer. The input bits are pointed to by bitsPtr and there are numBits of them. The first bit to decode is the most significant bit of the byte pointed to by bitsPtr. TheSymTable and numSyms are the same parameters that were passed to HuffmanDecodeInit.

TWO MONTHS AGO WINNER

Wow! The competition was really tight for the Erase Scribble Challenge. So tight, in fact, that the 5th place winner was only about 2% slower than the first place winner. But Challenge champion **Bob Boonstra** (Westford, MA) was able to implement code that was just a tiny bit more efficient than many other highly efficient entries. And, as a bonus, his entry was smaller than all but one of the other entries. Despite his post-publication disqualification from the Factoring Challenge (see below) Bob

THE RULES

Here's how it works: Each month we present a different programming challenge here. First, you write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to MacTech Magazine. We choose a winner based on code correctness, speed, size and elegance (in that order of importance) as well as the postmark of the answer. In the event of multiple equally-desirable solutions, we'll choose one winner at random (with honorable mention, but no prize, given to the runners up). The prize for each month's best solution is \$50 and a limited-edition "The Winner! MacTech Magazine Programming Challenge" T-shirt (not available in stores).

To help us make fair comparisons, all solutions must be in ANSI compatible C (e.g. don't use Think's Object extensions). Use only pure C code. We disqualify any entries with any assembly in them (except for challenges specifically stated to be in assembly). You may call any routine in the Macintosh toolbox (e.g., it doesn't matter if you use NewPtr instead of malloc). We test entries with the FPU and 68020 flags turned off in THINK C. We time

routines with the latest THINK C (with "ANSI Settings", "Honor 'register' first", and "Use Global Optimizer" turned on), so beware if you optimize for a different C compiler. **Limit your code to 60 characters wide.** This helps with e-mail gateways and page layout.

We publish the solution and winners for this month's Programmers' Challenge two months later. All submissions must be **received by** the 10th day of the month printed on the front of this issue.

Mark solutions "Attn: Programmers' Challenge Solution" and send them via e-mail - Internetprogchallenge@xplain.com, [AppleLink MT.PROGCHAL](mailto:AppleLinkMT.PROGCHAL), [CompuServe 71552,174](mailto:CompuServe71552,174) and [America Online MT.PRCHAL](mailto:AmericaOnlineMT.PRCHAL). Include the solution, all related files, and your contact info. If you send via snail mail, send a disk with those items on it; see "How to Contact Us" on p. 2.

MacTech Magazine reserves the right to publish any solution entered in the Programming Challenge of the Month. Authors grant MacTech Magazine the non-exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

remains our champion with four 1st place showings (including this one). Congratulations!

Here are the times and code sizes for each entry. Numbers in parens after a person's name indicate how many times that person has finished in the top 5 places of all previous Programmer Challenges, not including this one:

Name	time	code+data
Bob Boonstra (11)	1363	598
Ernst Munter (3)	1378	1434
John Schlack	1391	1482
Tom Elwertowski (1)	1394	910
Mark Chavira	1395	1538
Jim Sokoloff	1481	1094
Allen Stenger (7)	1911	988
Marcel Rivard	2233	2048
Joshua Glazer	168100	466

At least one contestant pointed out that this Challenge was not entirely realistic because: (1) in a real eraser situation the hit-test routine would return the point that was hit to the caller and, (2) the caller would be removing points from the scribble as segments were erased, thus removing the 'completely static scribble' characteristic of this Challenge. I agree, it would have been more realistic to have a dynamic scribble but I was trying to limit the complexity of the routine (and I was also trying to give clever people a chance to exploit the static nature of the data by using the init routine).

Bob's routine is well commented so I won't discuss it here. He chose an almost identical algorithm to everyone else but he implemented it just a touch better than everyone else.

NEW FACTORING WINNER

It seems that **Bob Boonstra** and I both made mistakes during the Factoring Challenge (June 1994 MacTech): Bob made the mistake of incorrectly handling some input values and I made the mistake of not finding them. Many thanks to **Jim Lloyd** (Mountain View, CA) for finding this bug and narrowing down the set of inputs that make it happen.

The bug only happens if the number to be factored was created from composite primes where one of them has the high bit set and the other one doesn't. If they're both set or if they're both clear then the bug doesn't happen. In case you're using Bob's code, there is a simple fix (thanks, Bob). Change this line:

```
*prime2Ptr = (x+y)>>1;
```

to this:

```
*prime2Ptr = (x>>1) + (y>>1) + 1;
```

Because of this bug I'm going to have to retro-actively disqualify Bob's entry from the Challenge and declare a new winner. However, the new winner is not simply the previous 2nd place winner. The new winner is from a guy whose code was originally sent in a day late (I had to disqualify it) but whose

performance is so much better than anyone else who entered (including Bob) that I'm going to allow it to win in the interest of having the best possible factoring code published. It's about two orders of magnitude faster than the other entries.

So, our new winner is **Nick Burgoyne** (Berkeley, CA). It turns out that this Challenge was right up Nick's alley, considering that he has taught factoring math classes in the past. His entry was the only one to use the quadratic sieve algorithm. If you're interested in learning more about this algorithm then Nick recommends a book by David Bressoud, *Factorization and Primality Testing*, published by Springer-Verlag in 1989. It covers the underlying mathematics and also gives further references to work on the quadratic sieve. It does not assume an advanced background in math. Nick is willing to discuss factoring with anyone who is interested via e-mail. His internet address is: sbrb@cats.ucsc.edu. Congrats, Nick!

Here's Bob's winning solution to the Erase Scribble Challenge, followed by Nick's winning solution to the Factoring Challenge:

SCRIBBLE.C

/* EraseScribble Copyright (c) 1994 J Robert Boonstra

Problem statement: Determine whether a square eraser of diameter eraserSize centered at the thePoint intersects any of the points in the data structure theScribble. Note that while the problem statement refers to line segments, the definition of a "hit" means that only the endpoints matter.

Solution strategy: The ideal approach would be to create a simple bitMap during Initialization indicating whether an eraser at a given location intersected theScribble. The bitMap would be created by stamping a cursor image at the location of each point in theScribble. However, a bitMap covering the required maximum bounding box of 1024 x 1024 would require 2^17 bytes, or four times as much storage as the 32K we are allowed to use.

Therefore, this solution has three cases:

- 1) If the actual bounding box for theScribble passed to the init function fits in the 32K available, we create a bitMap as above and use it directly.
 - 2) Otherwise, we attempt to create a half-scale bitMap, where each bit represents 4 pixels in the image, 2 in .h and 2 in .v. In the PtnInScribble function, we ca then quickly determine in most cases when the eraser does not intersect theScribble, and we have to walk the points in theScribble if the bitMap indicates a possible hit.
 - 3) In the event there is not enough storage for a half-scale bitmap, we create a quarter-scale bitmap, where each bit represents 16 pixels in the image, 4 in .h and 4 in .v. Then we proceed as in case 2.
- To optimize examination of the points in theScribble when the bitmap is not full scale, we sort the points in the initialization function and store them in the privateScribbleDataPtr. Although this reduces the amount of storage available for the bitmap by ~2K, it improves worst case performance significantly.

Although the init function is not timed for score, we have written it in assembler, in the spirit of the September Challenge.

*/

```
#pragma options(assign_registers,honor_register,mc68020)
```

```
#define ushort unsigned short
#define ulong unsigned long
#define kTotalStorage 0x8000
/*
```

Typedefs, defines, and prototypes

```
* Layout of privateScribbleData storage:
* OFFSET
```

CONTENT

```
* 0: bitMap
* kTotalStorage-kGlobals-4*gNumPoints: sorted points
* kTotalStorage-kGlobals: global data
```

```
* Globals stored in PrivateScribbleDataPtr
```

```
* gNumPoints: number of points in the scribble
* gHOrigin: min scribble h - eraserSize/2
* gVOrigin: min scribble v - eraserSize/2
* gBMHeight: max scribble h - min scribble h + eraserSize
* gBMWidth: max scribble v - min scribble v + eraserSize
```



```

*   gRowBytes: bitMap rowBytes
*   gMode:      flag indicating bitmap scale
*/
#define gNumPoints      kTotalStorage-2
#define gHOrigin        kTotalStorage-4
#define gVOrigin        kTotalStorage-6
#define gBMHeight       kTotalStorage-8
#define gBMWidth        kTotalStorage-10
#define gRowBytes       kTotalStorage-12
#define gMode           kTotalStorage-14
#define kGlobals         14
#define kSortedPoints   kTotalStorage-kGlobals
#define kBitMap          0
#define kHalfScaleBitMap 1
#define kQrtrScaleBitMap -1

typedef struct Scribble {
    Point startingPoint;
    Point deltaPoints[1];
} Scribble, *ScribblePtr, **ScribbleHndl;

void *EraseScribbleInit(ScribbleHndl theScribble,
    unsigned short eraserSize);

Boolean PtInScribble(Point thePoint,
    ScribbleHndl theScribble,
    unsigned short eraserSize,
    void *privateScribbleDataPtr);

PtInScribble

#define eraserH          D0
#define eraserV          D1
#define eraserSz         D2
#define pointCt          D6
#define scribblePt       D7

Boolean PtInScribble(Point thePoint,
    ScribbleHndl theScribble,
    unsigned short eraserSize,
    void *privateScribbleDataPtr)
{
    asm 68020 {
        MOVEM.L    D6-D7, -(A7)
        MOVE.L     thePoint, D1
        MOVEA.L     privateScribbleDataPtr, A0
        MOVEQ      #0, D0      ; clear high bits for BFTST
        MOVE.W     D1, D0
        SWAP       D1

; Return noHit if eraser is outside bounding box defined by gVOrigin, gHOrigin,
; gVOrigin+gBMHeight, gHOrigin+gBMWidth. Note that the bounding box has already
; been expanded by eraserSize/2 in each direction.
        SUB.W      gVOrigin(A0), D1
        BLT        @noHit      ; v < boundingBox.top
        CMP.W      gBMHeight(A0), D1
        BGT        @noHit      ; v > boundingBox.bottom
        SUB.W      gHOrigin(A0), D0
        BLT        @noHit      ; h < boundingBox.left
        CMP.W      gBMWidth(A0), D0
        BGT        @noHit      ; h > boundingBox.right

; Check eraser against bitMap; return noHit if not set
        MOVE.W     gRowBytes(A0), D2
        TST.W      gMode(A0)
        BNE.S      @testQrtrScaleBitMap

; Full-scale bitMap case; can return Hit if bit is set
        MULU.W     D1, D2      ; multiple row by rowBytes
        ADDA       D2, A0      ; A0 points to correct bitMap row
        BFTST      (A0)(D0:1)  ; D0 contains bit offset
        BNE        @hit

noHit:
        MOVEQ      #0, D0
        MOVEM.L    (A7)+, D6-D7
        UNLK       A6          ; save one branch by returning directly
        RTS

testQrtrScaleBitMap:
        BGT        @testHalfScaleBitMap

; Qrtr-scale bitMap case; cannot always return Hit if set
        LSR.W      #2, D1      ; * adjust for qrtr-scale bitMap *
        LSR.W      #2, D0      ; * adjust for qrtr-scale bitMap *
        MULU.W     D1, D2      ; multiple row by rowBytes
        ADDA       D2, A0      ; A0 points to correct bitMap row
        BFTST      (A0)(D0:1)
        BEQ        @noHit
        BRA.S      @TestPoints

testHalfScaleBitMap:
; Half-scale bitMap case; cannot always return Hit if set
        LSR.W      #1, D1      ; * adjust for half-scale bitMap *
        LSR.W      #1, D0      ; * adjust for half-scale bitMap *

```

Developers:

PatchWorks™

Builds Updaters Without Programming

PatchWorks has many options, but only one function: to create updater applications for distribution to end users over channels that may be non-secure (e.g., BBSs).

Before the advent of **PatchWorks**, creating an updater was a project in itself, one that consumed valuable programmer time which could more profitably be spent on revenue-producing projects.

With **PatchWorks**, you create an updater in minutes. Since there's no coding or scripting, no bugs are introduced. Just fill in a dialog, and **PatchWorks** does the rest!

Distribute updaters frequently to reflect maintenance releases, and watch your tech support and fulfillment costs fall dramatically.

Most important, your customers will know you care.

PW Settings			
Old File...	Offline™ 2.1 (APPL, OFFL)		
New File...	Offline™ 2.2 (APPL, OFFL)		
Updater...	Offline Updater		
Log file...	PW Log - Offline™ 2.2		
Compression		High	<input checked="" type="radio"/> Rewrite updater <input type="radio"/> Merge resources into updater
Old File {	Name	Any	
	Type	Same as above	<input type="checkbox"/> Data fork
	Creator	Same as above	
	Screen Name	Specify =>	Offline
New File {	Name	Specify =>	Offline™
	Mod Date	Date patched	<input type="checkbox"/> Copy all

Features

- Works with apps, INITs, cdevs, fonts, drivers, etc.
- Updaters support up to 8 old versions
- Resource compression (diffing) produces small updaters
- Resource encryption makes updaters hacker-resistant
- Preserves personalization data (name, serial #, etc.)
- Updater distribution is **unrestricted & royalty-free**

Pricing: Begins at \$195. Call for more information.



SNA, Inc.
 2200 NW Corporate Blvd.
 Boca Raton, FL 33431
 Tel (407) 241-0308 • FAX (407) 241-3195


```

    MULU.W    D1,D2      ; multiple row by rowBytes
    ADDA.L    D2,A0      ; A0 points to correct bitMap row
    BETST     (A0){D0:1}
    BEQ       @noHit
TestPoints:
; bitMap indicates there might be a hit, need to check
    MOVEA.L   privateScribbleDataPtr,A0
    MOVE.W    eraserSize,eraserSz
    MOVE.L    thePoint,eraserH
    MOVE.L    eraserH,eraserV
    SWAP      eraserV
    LEA       kSortedPoints(A0),A1
; Scan sets of 64 points to find a close match
    MOVE.W    gNumPoints(A0),D7
    LSR.W     #6,D7      ; numPoints/64
    MOVE.W    D7,pointCt
    LSL.W     #8,D7      ; times 4 bytes/pt * 64 pts
    SUBA.L    d7,A1
    SUBQ      #1,pointCt
eightLoop:
    MOVE.L    -(A1),scribblePt
    CMP.W     eraserH,scribblePt
    BLT.S     @hLoop
    ADDI      #65*4,A1
    DBRA      pointCt,@eightLoop;
; Note that the sorted scribblePts have been stored with
; eraserSize/2 already added in h and v
hLoop:
    MOVE.L    -(A1),scribblePt
    CMP.W     eraserH,scribblePt
    BLT.S     @hLoop
; All points from here on have a true .h component >= eraserH-eraserSize/2. Now
; need to look at those where .h <= eraserH+eraserSize/2. Because we already added
; eraserSize/2 to the sorted point values, we compare against eraserH+eraserSz
    ADD.W     eraserSz,eraserH
    ADD.W     eraserV,eraserSz
vLoop:
    CMP.W     eraserH,scribblePt
    BGT.S     @noHit
; scribblePt.h is in range, now check .v
    SWAP      scribblePt
    CMP.W     eraserV,scribblePt
    BLT.S     @vLoopCheck
    SUB.W     eraserSz,scribblePt
    BLE.S     @hit
vLoopCheck:
    MOVE.L    -(A1),scribblePt
    BRA       @vLoop;
hit:
    MOVEQ     #1,D0
    MOVEM.L   (A7)+,D6-D7
}

void *EraseScribbleInit(ScribbleHndl theScribble,
    unsigned short eraserSize)
{
    ulong bitMapStorageAvail;
    asm 68020 {
        MOVEM.L   D3-D7/A2,-(A7)
; Allocate storage - use full 32K.
; Note that it is the responsibility of the caller to release this storage.
        MOVE.L    #kTotalStorage,D0
        NewPtr     CLEAR
        MOVE.L    A0,A2      ; A0 is privateDataPtr
; Scan thru the scribble to find min and max in h and v
        MOVEA.L   theScribble,A1
        MOVEA.L   (A1),A2    ; A2 = *theScribble
; Initialize mins and maxes to be the starting point
        MOVE.L    (A2)+,D7    ; current scribble point
        MOVEQ     #0,D5      ; clear high bits for ADDA.L later
        MOVE.W    D7,D5      ; D5 = max h
        MOVE.W    D7,D4      ; D4 = min h
        MOVE.W    D7,D1      ; D1 = current h
        SWAP      D7          ; D7 = max v
        MOVE.W    D7,D6      ; D6 = min
        MOVE.W    D7,D2      ; D2 = current v
        LEA       kSortedPoints(A0),A1 ; sorted point storage
; Set up eraserSize/2
        MOVE.W    eraserSize,D3
        LSR.W     #1,D3      ; D3 = eraserSize/2
minMaxLoop:
; Store point for subsequent sorting
        MOVE.W    D1,D0

```

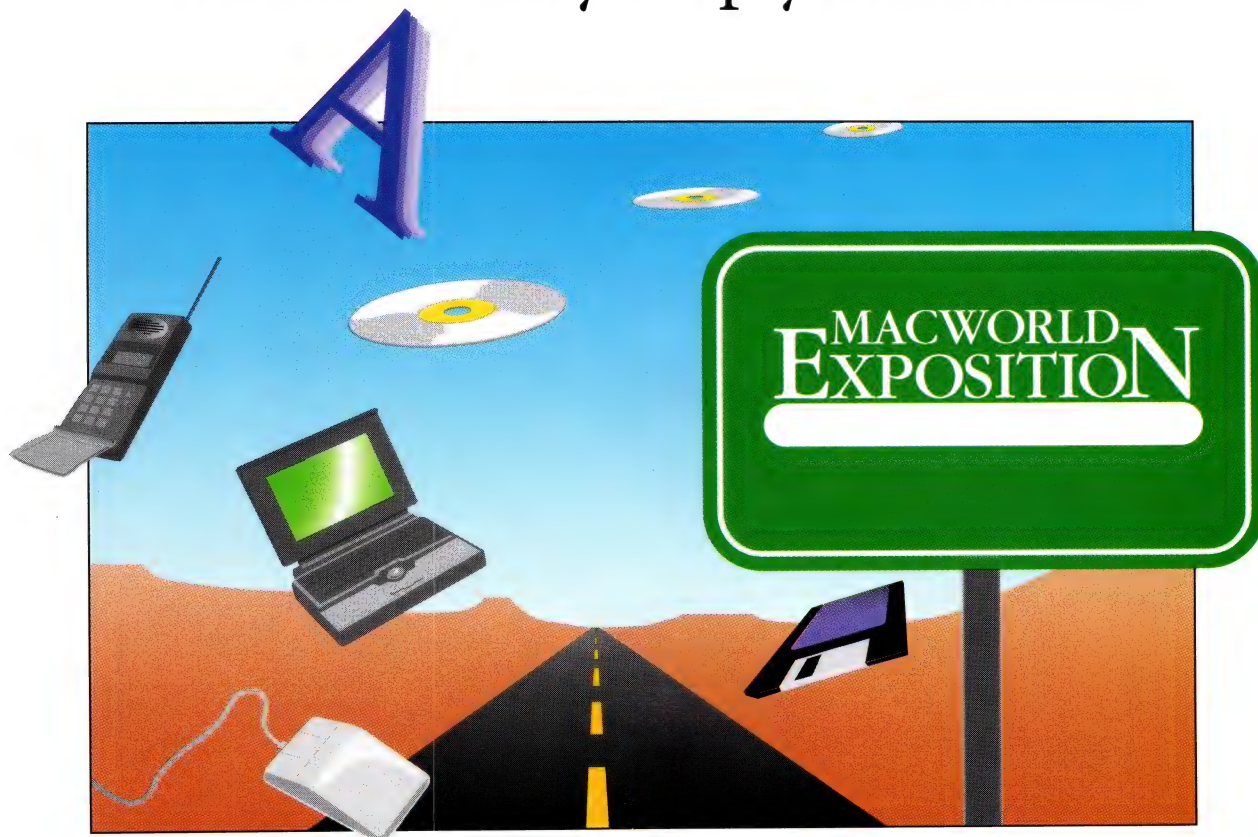
```

        ADD.W     D3,D0
        MOVE.W    D0,-(A1) ; store current h + eraserSize/2
        MOVE.W    D2,D0
        ADD.W     D3,D0
        MOVE.W    D0,-(A1) ; store current v + eraserSize/2
; Fetch next point
        MOVE.L    (A2)+,D0 ; fetch deltaPoint
        BEQ       @minMaxDone
        ADD.W     D0,D1      ; update current h
        CMP.W     D1,D5
        BGE       @noNewHMax
        MOVE.W    D1,D5      ; store new max h
        BRA.S     @noNewHMin
noNewHMax:
        CMP.W     D1,D4
        BLE       @noNewHMin
        MOVE.W    D1,D4      ; store new min h
noNewHMin:
        SWAP      D0
        ADD.W     D0,D2      ; update current v
        CMP.W     D2,D7
        BGE       @noNewVMax
        MOVE.W    D2,D7      ; store new max v
        BRA.S     @noNewVMin
noNewVMax:
        CMP.W     D2,D6
        BLE       @noNewVMin
        MOVE.W    D2,D6      ; store new min v
noNewVMin:
        BRA.S     @minMaxLoop
minMaxDone:
; Calculate number of points
        LEA       kSortedPoints(A0),A2
        MOVE.L    A2,D0
        SUB.L     A1,D0
        LSR.W     #2,D0
        MOVE.W    D0,gNumPoints(A0)
; Calculate bitmap storage available
        SUBQ.L    #8,A1      ; Reserve room for sentinals
        MOVE.L    A1,D0
        SUB.L     A0,D0
        MOVE.L    D0,bitMapStorageAvail
; Calculate origin = min h/v minus eraserSize/2
        MOVE.W    eraserSize,D0
; Calculate number of columns
        SUB.W     D3,D4      ; adjust h origin for eraserSize/2
        MOVE.W    D4,gH0origin(A0) ; D4 = H Origin
        ADD.W     D0,D5      ; add eraserSize to width
        SUB.W     D4,D5
        MOVE.W    D5,gBMWidth(A0)
        ADDQ      #7,D5      ; round up to byte level
        LSR.W     #3,D5
        MOVE.W    D5,gRowBytes(A0) ; D5 = rowBytes
; Calculate number of rows
        SUB.W     D3,D6      ; adjust v origin for eraserSize/2
        MOVE.W    D6,gV0origin(A0) ; D6 = V Origin
        ADD.W     D0,D7      ; add eraserSize to height
        SUB.W     D6,D7
        MOVE.W    D7,gBMHeight(A0) ; D7 = number of rows
        ADDQ      #1,D7
; Calculate number of bytes needed to store bitmap.
        MULU.W    D5,D7
        CMP.L     bitMapStorageAvail,D7
        BLE.S     @haveEnoughStorage
; Not enough storage, so we try a half-scale bitMap
        MOVEQ     #kHalfscaleBitMap,D1
        MOVE.W    D1,gMode(A0)
; Adjust gRowBytes for half-scale bitMap
        MOVE.W    gBMWidth(A0),D5
        ADDI      #15,D5      ; round up to byte level
        LSR.W     #4,D5
        MOVE.W    D5,gRowBytes(A0) ; D5 = rowBytes
        LSR.W     #1,D0      ; adjust eraser size for half-scale bitmap
        MOVE.W    gBMHeight(A0),D7
        ADDQ      #1,D7
        LSR.W     #1,D7
        ADDQ      #1,D7
        MULU.W    D5,D7
        CMP.L     bitMapStorageAvail,D7
        BLE.S     @haveEnoughStorage
; Still not enough storage, so we set up a qtr-scale bitMap
        MOVEQ     #kQtrscaleBitMap,D1

```


Forget Gas, Food & Lodging

On the Information Superhighway
this is the only stop you'll need.



Don't want to be bypassed on the Information Superhighway? Then plan a detour to MACWORLD Expo. Here you'll test drive the products and services that enable you to maximize the potential of the Macintosh now and down the road.

MACWORLD Expo is your chance to see hundreds of companies presenting the latest in turbo-charged Macintosh technology. Make side by side comparisons of thousands of Macintosh products. Learn from the experts how to fine-tune your system and what products will keep your engine running smooth. Attend a variety of information-packed conference programs that provide the skills and knowledge to put you in the driver's seat. So pull on in and take that new Mac for a spin.

Add a stop at any of our upcoming MACWORLD Expo events to your information roadmap. With shows in **San Francisco, Boston and Toronto**, we're just around the next bend.

Please send me more information on MACWORLD Expo.

I am interested in: ☐ Exhibiting ☐ Attending

☐ **San Francisco** ☐ **Boston** ☐ **Toronto**

Name _____

Title _____

Company _____

Address _____

City/State/Zip _____

Phone _____ Fax _____

Mail to: Mitch Hall Associates, 260 Milton Street, Dedham, MA 02026

Or Fax to: 617-361-3389 Phone: 617-361-8000


```

MOVE.W D1,gMode(A0)
; Adjust gRowBytes for qrtr-scale bitMap
MOVE.W gBMWidth(A0),D5
ADDI #31,D5 ; round up to byte level
LSR.W #5,D5
MOVE.W D5,gRowBytes(A0) ; D5=rowBytes
LSR.W #1,D0 ; adjust eraser size for qrtr-scale bitmap

```

haveEnoughStorage:

; Create bitMap

```

MOVEA.L theScribble,A1
MOVEA.L (A1),A2

```

; Initial location to stamp eraser image in bitmap is the

; startingPoint, minus the bitmap origin, minus eraserSize/2;

```

MOVE.L (A2)+,D2 ; Fetch startingPoint
MOVE.W D2,D1 ; D1 = current scribble point h
SWAP D2 ; D2 = current scribble point v
SUB.W D4,D1 ; D1 = scribble point h - H origin
SUB.W D3,D1 ; offset h by -eraserSize/2
SUB.W D6,D2 ; D2 = scribble point v - V origin
SUB.W D3,D2 ; offset v by -eraserSize/2
MOVEQ #0,D4 ; clear high bits for BFINS later

```

```

MOVE.W D0,D7 ; save eraser width

```

```

MOVEQ #-1,D3 ; set bits for insertion using BFINS;

```

; Loop through all points in the scribble

stampPointLoop:

```

MOVEA.L A0,A1
MOVE D7,D6 ; D6 = row counter for DBRA smear
MOVE.W D2,D0 ; D0 = v - origin

```

```

TST.W gMode(A0)
BEQ @L1
BGT @L0

```

```

LSR.W #1,D0 ; adjust for qrtr-scale bitmap

```

```

L0: LSR.W #1,D0 ; adjust for half-scale bitmap

```

L1:

```

MULU.W D5,D0 ; D0 = (v - origin) * rowBytes
ADDA.L D0,A1 ; A1 = privateStorage - rowOffset
MOVE.W D7,D0
ADDQ #1,D0 ; D0=eraserSize/2+1, the number of
; bits to set in bitMap;

```

; Loop through eraserSize+1 rows in bitMap for this point

stampRowLoop:

```

MOVE.W D1,D4 ; D4 = scribble h - origin

```

```

TST.W gMode(A0)
BEQ @L3
BGT @L2

```

```

LSR.W #1,D4 ; adjust for qrtr-scale bitmap

```

```

L2: LSR.W #1,D4 ; adjust for half-scale bitmap

```

L3:

```

BFINS D3,(A1){D4:D0};insert mask D3 of width D0
; into bitmap A1 at offset D4
ADDA.L D5,A1 ; increment by rowBytes
DBRA D6,@stampRowLoop

```

; Fetch next deltaPoint, quit if done

```

MOVE.L (A2)+,D0
BEQ @bitMapDone
ADD.W D0,D1 ; update current scribble h
SWAP D0
ADD.W D0,D2 ; update current scribble v
BRA.S @stampPointLoop

```

bitMapDone:

; Sort scribble points for fast lookup. Simple exchange sort will do.

outerSortLoop:

```

MOVEQ #0,D6 ; exchange flag = no exchanges
LEA kSortedPoints(A0),A2 ; sorted pt storage
MOVE.W gNumPoints(A0),D7 ; outer loop counter
SUBQ #2,D7 ; DBRA adjustment
MOVE.L -(A2),D0 ; D0 = compare value - h
MOVE.L D0,D1
SWAP D1 ; D1 = compare value - v

```

innerSortLoop:

```

MOVE.L -(A2),D2
MOVE.L D2,D3
SWAP D3
CMP.W D0,D2 ; primary sort by h
BLT.S @doSwap
BGT.S @noSwap
CMP.W D1,D3 ; secondary sort by v

```

```

BGE.S @noSwap
doSwap:
MOVE.L D2,D(A2)
MOVE.L D0,(A2)
MOVEQ #1,D6
BRA.S @testInnerLoop

```

noSwap:

```

MOVE.L D2,D0
MOVE.W D3,D1

```

testInnerLoop:

```

DBRA D7,@innerSortLoop
TST.W D6
BNE.S @outerSortLoop;
MOVE.L #0x80008000,-(a2) ; Sentinel
; to end point loop
MOVE.L #0x7FFF7FFF,-(a2) ; Sentinel
; to end point loop

```

```

MOVE.L A0,D0 ; return ptr to private data
MOVEM.L (A7)+,D3-D7/A2

```

NICK'S FACTORING SOLUTION

Factor64.c

// Factor the product N of two primes each ≤ 32 bits
#include "Factor64.h"

// Factor64() is a simplified version of the quadratic
// sieve using multiple polynomials

```

void Factor64(ulong nL,ulong nH,
              ulong *p1,ulong *p2) {

```

```

    register ushort ls,k;
    register ulong i,j;

```

```

    ushort p,s,qi,hi,r,c,sP,sN,ts;
    ushort b,m,bm,br,m2,S[5];
    ulong d,e,sX,sQ,sq;

```

```

    Int B,C,Q,R,T,X,Y;
    ushort *Ptr,*pf,*sf,*lg,*r1,*r2;
    ushort *lv,**hm,*hp,**gm,*gp,*gi;
    ushort *pv,**Xv,*Yv;

```

// Check whether N is square

```

N[1] = nL & 0xffff; N[2] = nL >> 16;
N[3] = nH & 0xffff; N[4] = nH >> 16;
k = 4; while (N[k] == 0) k++; N[0] = k;

```

```
sq = floor(sqrt(nL + 4294967296.0*nH));
```

```
Set(S,sq); Mul(S,S,S);
```

```

if (Comp(S,N) == 0) {
    *p1 = sq;
    *p2 = sq;
    return;
}

```

// Check N for small factors (up to 0x800)

```

for (k = 0; k < 616; k += 2) {
    p = Prm[k];
    if (Modq(N,p) == 0) {
        Divq(N,p);
        *p1 = p;
        *p2 = Unset(N);
        return;
    }
}

```

// Allocate memory

```

Ptr = malloc(0x20000);
if (Ptr == 0) {
    printf(" malloc failed \n");
    exit(0);
}

```

```

X = (ushort *) Ptr; Ptr += 20;
Y = (ushort *) Ptr; Ptr += 20;
B = (ushort *) Ptr; Ptr += 20;
C = (ushort *) Ptr; Ptr += 20;
Q = (ushort *) Ptr; Ptr += 20;
R = (ushort *) Ptr; Ptr += 20;
T = (ushort *) Ptr; Ptr += 20;
pf = (ushort *) Ptr; Ptr += 120;
sf = (ushort *) Ptr; Ptr += 120;
lg = (ushort *) Ptr; Ptr += 120;

```

```

r1 = (ushort *) Ptr; Ptr += 120;
r2 = (ushort *) Ptr; Ptr += 120;
lv = (ushort *) Ptr; Ptr += 12000;
hm = (ushort **) Ptr; Ptr += 240;
hm[0] = (ushort *) Ptr; Ptr += 14400;
gm = (ushort **) Ptr; Ptr += 240;
gm[0] = (ushort *) Ptr; Ptr += 24000;
pv = (ushort *) Ptr; Ptr += 120;
Yv = (ushort *) Ptr; Ptr += 120;
Xv = (ushort **) Ptr; Ptr += 240;
Xv[0] = (ushort *) Ptr;

```

```

for (k = 1; k < 120; k++) {
    hm[k] = hm[k-1] + 120;
    gm[k] = gm[k-1] + 120 + k;
    Xv[k] = Xv[k-1] + 6;
}

```

// sieve parameters: ts and qs are cutoffs, b is size of
prime base and m2 is size of sieve

```

k = Bitsize(N);
b = 2 + (5*k)/4;
bm = b + 3;

```

```

if (k > 40) m = 50*k + 400;
else m = 100*k - 1600;
m2 = m + m;

```

```

if (k > 40) ts = 196 + k/2;
else ts = 11*k - 24 - (k*k)/8;
sq = ceil(sqrt(sq/m));

```

// Construct the prime base

```

L0: k = 2;
i = 0;
while (k < b) {
    p = Prm[i];
    i++;
    s = Modq(N,p);
    if (qrs(s,p) == 1) {
        pf[k] = p;
        sf[k] = mrt(s,p);
        lg[k] = Prm[i];
        k++;
    }
    i++;
}
pf[1] = 2;

```

// Construct quadratic polynomials as needed

```

while (Prm[i] < sq) i += 2;
hi = i;
qi = 0;

```

```

L1: do {if (hi < 616) sP = Prm[hi];
      else sP = npr(sP);
      while ((sP&3) == 1) {
          hi += 2;
          if (hi < 616) sP = Prm[hi];
          else sP = npr(sP);
      }
      sN = Modq(N,sP);
      hi += 2;
    } while (qrs(sN,sP) == -1);

```

```

Set(T,sN);
Dif(T,T,N);
Divq(T,sP);
d = Modq(T,sP); d *= sP; d += sN;
sX = hrt(d,sP);
Set(X,sX);

```

```

e = (ulong) sP*sP;
Set(B,e);
Mulq(B,m); Dif(B,B,X);
Mul(C,B,B); Dif(C,C,N);
Divq(C,sP); Divq(C,sP);

```

// Do the sieve for current polynoi

```

if ((B[1] & 1) == 0) {i = 1; j = 0;}
else {i = 0; j = 1;}

```

```

if ((N[1] & 7) == 0) ls = 21;
else if ((N[1] & 3) == 0) ls = 14;
else ls = 7;

```

```
while (i < m2) {
```



```

    lv[i] = ls; i += 2;
    lv[j] = 0; j += 2;
}

for (k = 2; k < b; k++) {
    p = pf[k];
    s = sf[k];
    e = sX % p;
    d = sP % p; d *= d; d %= p; d = inv(d,p);

    if (e < s) e += p;
    i = e-s; i *= d; i %= p; i = m-i; i %= p;
    r1[k] = i;

    j = e+s; j *= d; j %= p; j = m-j; j %= p;
    r2[k] = j;

    if (i > j) {
        e = i;
        i = j;
        j = e;
    }
    ls = lg[k];
    while (j < m2) {
        lv[i] += ls; i += p;
        lv[j] += ls; j += p;
    }
    if (i < m2) lv[i] += ls;
}

// Factor polynomial to find rows of matrix hm[qi,k]
for (i = 0; i < m2; i++)
    if (lv[i] > ts) {
        hp = hm[qi];
        d = (ulong) i*sP;
        Set(T,d); Mul(Q,T,T); Add(Q,Q,C);
        R[0] = B[0]; R[1] = B[1];
        R[2] = B[2]; R[3] = B[3];
        s = i+i; Mulq(R,s);
        if (Comp(Q,R) == 1) hp[0] = 0;
        else hp[0] = 1;
        Dif(Q,Q,R);

        hp[1] = 0;
        while ((Q[1] & 1) == 0) {
            hp[1] += 1;
            Shiftr(Q);
        }

        k = b;
        while (Q[0] > 2) {
            k--;
            if (k == 1) goto L2;
            p = pf[k];
            j = i % p;
            if (j == r1[k] || j == r2[k]) {
                hp[k] = 1;
                Divq(Q,p);
                while (Modq(Q,p) == 0) {
                    hp[k] += 1;
                    Divq(Q,p);
                }
            }
            else hp[k] = 0;
        }
        sQ = Unset(Q);
        while (sQ > 1) {
            k--;
            if (k == 1) goto L2;
            p = pf[k];
            j = i % p;
            if (j == r1[k] || j == r2[k]) {
                hp[k] = 1;
                sQ /= p;
                while (sQ%p == 0) {
                    hp[k] += 1;
                    sQ /= p;
                }
            }
            else hp[k] = 0;
        }
        while (k > 2) {
            k--;
            hp[k] = 0;
        }
        Mulq(T,sP);
    }
}

Dif(Xv[qi],T,B);
Yv[qi] = sP;
qi += 1;
if (qi == bm) goto L3;
}
goto L1;

// Row reduce gm = hm % 2 and find X^2 = Y^2
// (mod N)
L3: k = N[0];
g = (double) N[k];
if (d > 1) g += N[k-1]/65536.0;
if (d > 2) g += N[k-2]/4294967296.0;
if (d > 3) g += 1/4294967296.0;

for (r = 0; r < bm; r++) {
    hp = hm[r]; gp = gm[r];
    for (c = 0; c < b; c++)
        gp[c] = hp[c] & 1;
    br = b + r;
    for (c = b; c < br; c++) gp[c] = 0;
    gp[br] = 1;
}

for (r = 0; r < bm; r++) {
    br = b + r;
    gp = gm[r];
    c = b - 1;
    while (gp[c] == 0 && c > 0) c--;

    if (c > 0 || gp[0] == 1) {
        for (i = r+1; i < bm; i++) {
            gi = gm[i];
            if (gi[c] == 1) {
                gi[c] = 0;
                for (k = 0; k < c; k++)
                    gi[k] ^= gp[k];
                for (k=b; k<br; k++)
                    gi[k] ^= gp[k];
            }
        }
    }
    else {
        for (i = 1; i < b; i++)
            pv[i] = 0;
        Set(X,1); Set(Y,1);
        for (k = b; k <= br; k++)
            if (gp[k] == 1) {
                j = k - b;
                Mul(X,X,Xv[j]);
                if (X[0]>12) Mod(X);
                Mulq(Y,Yv[j]);
                if (Y[0]>12) Mod(Y);
                for (i=1; i<b; i++)
                    pv[i] += (long) hm[j][i];
            }
        Mod(X);

        k = pv[1] >> 1;
        while (k > 15) {
            for (i = Y[0]; i > 0; i--)
                Y[i+1] = Y[i];
            Y[1] = 0;
            Y[0] += 1;
            k -= 16;
            if (Y[0] > 12) Mod(Y);
        }
        Mulq(Y,1 << k);

        for (i = 2; i < b; i++) {
            j = pv[i];
            if (j == 0) continue;
            if (j == 2) Mulq(Y,pf[i]);
            else {
                T[1] = pf[i];
                T[0] = 1;
                while (j > 2) {
                    j -= 2;
                    Mulq(T,pf[i]);
                }
                Mul(Y,Y,T);
            }
            if (Y[0] > 12) Mod(Y);
        }
        Mod(Y);
        Dif(T,X,Y);
    }
}

Add(R,X,Y);
if (Comp(N,R)<=0) Dif(R,R,N);
if (T[0] == 0 || R[0] == 0)
    continue;
*p1 = Gcd(T);
*p2 = Gcd(R);
return;
}

bm += 5;
ts -= 2;
goto L0;
}

End of the function Factor64Q.

// Inverse of b modulo m (Euclid's algorithm)
ulong inv(ulong b,ushort m) {
    register ulong u,v,t,n;
    u = 1;
    v = 0;
    t = 1;
    n = m;

    for (;;) {
        while ((b & 1) == 0) {
            v <<= 1;
            if (t & 1) t += m;
            t >>= 1;
            b >>= 1;
        }
        if (b == 1) break;

        if (b > n) {
            b -= n;
            u += v;
        }
        else {
            n -= b;
            v += u;
        }
        while ((n & 1) == 0) {
            u <<= 1;
            if (t & 1) t += m;
            t >>= 1;
            n >>= 1;
        }
    }
    t *= u; t %= m;
    return t;
}

// Determine if n is square modulo p (QR algorithm)
short qrs(ushort n,ushort p) {
    register short j;
    register ushort n2,n4;

    if (n == 1) return 1;

    j = 1;
    for (;;) {
        n2 = n + n;
        n4 = n2 + n2;
        p %= n4;
        while (p > n) {
            if (n & 2) j = -j;
            if (p > n2) p -= n2;
            else p = n2 - p;
        }
        if (p == 1) return j;
        n %= p;
        if (n == 1) return j;
    }
}

// Square root of n modulo p
ushort mrt(ushort n,ushort p) {
    register ulong q,s;
    register long x,u,v;
    ulong m;
    long t,r;

```


It's Macintosh Accounting At Its Best



FLEXIBLE

FlexWare is designed to give you almost unlimited possibilities as your company grows.



POWERFUL

FlexWare's accounting features and reporting capabilities give you information you'll rely on daily to make smart business decisions.



FAST

Large businesses consistently chose FlexWare because of its exceptional speed on network and client/server systems.

Meet your clients' needs with the FlexWare Development System. FlexWare is an integrated, modular, MacWorld award-winning accounting program that you can modify and customize to suit each of your clients' requirements. Take advantage of FlexWare's top performance and adaptability. It's flexible. It's powerful. It's fast. Call Jack Robinson at extension 2309 for information on the FlexWare Development System.

FlexWare®

ACCOUNTING



2130 Professional Drive, Suite 150, Roseville, CA 95661-3751 • 916-781-3880 • FAX 916-781-3814 • 1-800-447-5700

```

if (n == 1) return 1;
q = (p+1) >> 1;

m = n;
if ((q & 1) == 0) {
    q >>= 1;
    s = 1;
    while (q > 0) {
        if (q & 1) {
            s *= m; s %= p;
        }
        m *= m; m %= p;
        q >>= 1;
    }
    if (s+s > p) s = p-s;
    return s;
}

s = 0;
t = n;
while (qrs(t,p) == 1) {
    s += 1;
    t += 1-s-s;
    if (t%p == 0) {
        if (s+s > p) s = p-s;
        return s;
    }
    if (t < p) t += p;
}

q >>= 1;
n = t;
r = s;
u = 1;
v = 1;
while (q > 0) {
    x = n*u; x %= p; x *= u; x %= p;
    t = r*r; t %= p;
    if (t >= x) x = t-x;
    else x = t-x+p;
    u *= r; u %= p; u += u; u %= p;
}
    
```

```

r = x;
if (q & 1) {
    x = n*u; x %= p; x *= v;
    x %= p;
    t = s*r; t %= p;
    if (t >= x) x = t-x;
    else x = t-x+p;
    v *= r; v %= p; v += s*u; v %= p;
    s = x;
}
q >>= 1;

if (s+s > p) s = p-s;
return s;

// Square root of n modulo p^2 for p = 3 (mod 4)
hrt

ulong hrt(ulong n, ushort p) {
    register ulong s, t, x, y;
    ulong m;

    m = n%p;
    s = m;
    y = 1;
    t = p-3; t >>= 2;
    while (t > 0) {
        if (t&1) {
            y *= s; y %= p;
        }
        s *= s; s %= p;
        t >>= 1;
    }
    x = m*y; x %= p;
    s = (ulong) p*p;
    t = x*x;
    if (n < t) n += s;
    n -= t; n /= p; n *= y; n %= p;
    t = p; t += 1; t >>= 1;
    n *= t; n %= p; n *= p; n += x;
    if (n+n > s) n = s-n;
}
    
```

```

return n;
}

// Get next prime
npr

ushort npr(ushort p) {
    register ushort d, s, k;

    do {p += 2;
        s = floor(sqrt(p));
        k = 0;
        d = 3;
        while (d <= s) {
            if (p%d == 0) break;
            k += 2;
            d = Prm[k];
        }
    } while (d <= s);
    return p;
}

// Addition S = A + B
Add

void Add(Int S, Int A, Int B) {
    register ushort *pH, *pL;
    register ulong t;
    register ushort c, k;
    ushort s, dH, dL;

    if (A[0] > B[0]) {
        pH = A; dH = A[0]; pL = B; dL = B[0];
    } else {
        pH = B; dH = B[0]; pL = A; dL = A[0];
    }
    if (dL == 0) {
    }
    }
    
```



```

if (S != pH)
    for (k=0;k<=dH; k++) S[k]=pH[k];
return;
}

k = 0;
c = 0;
while (k < dL) {
    k++;
    t = (ulong) pH[k] + pL[k] + c;
    if (t >= 0x10000) {
        t -= 0x10000;
        c = 1;
    }
    else c = 0;
    S[k] = t;
}
while (c == 1 && k < dH) {
    k++;
    s = pH[k];
    if (s == 0xFFFF) {
        S[k] = 0;
        c = 1;
    }
    else {
        S[k] = s + 1;
        c = 0;
    }
}
while (k < dH) {
    k++;
    S[k] = pH[k];
}
if (c == 1) {
    dH += 1;
    S[dH] = 1;
}
S[0] = dH;
}

```

// Difference D = |A-B|

```

void Dif(Int D,Int A,Int B) {
    register ushort *pH, *pL;
    register long t;
    register ushort c,k;
    ushort s,dH,dL;
    short e;

    k = A[0];
    if (k > B[0]) e = 1;
    else {
        if (k < B[0]) e = -1;
        else {
            while (A[k]==B[k] && k > 0) k--;
            if (k == 0) {
                D[0] = 0;
                return;
            }
            if (A[k] > B[k]) e = 1;
            else e = -1;
        }
    }
    if (e == 1) {
        pH = A; dH = A[0];
        pL = B; dL = B[0];
    }
    else {
        pH = B; dH = B[0];
        pL = A; dL = A[0];
    }
    if (dL == 0) {
        if (D != pH)
            for (k=0;k<=dH;k++) D[k]=pH[k];
        return;
    }

    c = 0;
    k = 0;
    while (k < dL) {
        k++;
        t = (long) pH[k] - pL[k] - c;
        if (t < 0) {
            t += 0x10000;
            c = 1;
        }
    }
}

```

Dif



Discover the database leading developers use to create award-winning software.

C-Index/II Database Library

Features

Fast B+Tree Access
Unlimited Key Types
PowerFail Protection
Extremely Flexible
Easy to Learn and Use
Live Cross-Platform
Compatibility with
Macintosh, Windows,
DOS, Unix, NT, etc.

JPL
Intuit
FDIC
PG&E
Stratus
Boeing
Banyan
Software
Toolworks
Symantec
Manzanita
Timberline
many more

Benefits

Faster Time To Market
Increased Reliability
No Runtime Royalties
Complete Source: \$695

Contact us for more information



Trio Systems

936 E. Green St. Suite 105
 Pasadena, CA 91106-2945
 818/584-9706
 818/584-0364 Fax
 mail@triosystems.com


```

    else c = 0;
    D[k] = t;
}
while (c == 1 && k < dH) {
    k++;
    s = pH[k];
    if (s == 0) {
        D[k] = 0xFFFF;
        c = 1;
    }
    else {
        D[k] = s - 1;
        c = 0;
    }
}
while (k < dH) {
    k++;
    D[k] = pH[k];
}
k = dH;
while (D[k] == 0 && k > 0) k--;
D[0] = k;
}

```

// Multiply P = A * B

void Mul(Int P, Int A, Int B) { Mul

```

    register ushort *pB;
    register ulong t;
    register ushort s, n, k;
    ushort d, j;

    if (A[0] == 0 || B[0] == 0) {
        P[0] = 0;
        return;
    }
    d = A[0] + B[0];
    for (k = 1; k <= d; k++) bufm[k] = 0;

```

```

    pB = B;
    j = 0;
    while (j < A[0]) {
        j++;
        s = A[j];
        k = 0;
        n = j;
        while (k < pB[0]) {
            k++;
            t = (ulong) s * pB[k];
            bufm[n] += t & 0xFFFF;
            n++;
            bufm[n] += t >> 16;
        }
    }

```

```

    k = 1;
    while (k < d) {
        t = bufm[k];
        bufm[k] = t & 0xFFFF;
        k++;
        bufm[k] += t >> 16;
    }
    if (bufm[d] == 0) d -= 1;

```

```

    for (k = 1; k <= d; k++) P[k] = bufm[k];
    P[0] = d;
}

```

// Quick multiply A = A * b

void Mulq(Int A, ushort b) { Mulq

```

    register ushort *pA;
    register ulong t, w, c;
    register ushort d, k;

    d = A[0];
    if (d == 0) return;

    pA = A;
    if (b == 0) {
        pA[0] = 0;
        return;
    }

```

```

    c = 0;
    k = 0;
    while (k < d) {
        k++;
        t = (ulong) pA[k] * b;
        w = (t & 0xFFFF) + c;
        c = t >> 16;
        if (w >= 0x10000) {
            w -= 0x10000;
            c += 1;
        }
        pA[k] = w;
    }
    if (c > 0) {
        d += 1;
        pA[d] = c;
    }
    A[0] = d;
}

```

// Modulo R = R % N

void Mod(Int R) { Mod

```

    register ulong t;
    register long w;
    register ushort k, j;
    ushort d, n, tL, tH;
    ulong c;
    short e;
    double f;

    d = N[0];
    n = R[0];
    if (d > n) return;

    for (k = 1; k <= n; k++) buf[k] = R[k];

    while (n >= d) {
        f = buf[n] * 65536.0;
        if (n > 1) f += buf[n-1];
        t = f/g;
        e = n - d;
        if (e > 0) {
            tL = t & 0xFFFF;
            tH = t >> 16;
        }
        else {
            tL = t >> 16;
            if (tL == 0) {
                if (t < 0xFFFF) break;
            }
            else {
                k = d;
                while (buf[k] == N[k]
                    && k > 0) k--;
                if (k > 0 && buf[k] < N[k])
                    break;
                tL = 1;
            }
        }
        tH = 0;
        e = 1;
    }

```

```

    c = 0;
    j = e;
    for (k = 1; k <= d; k++) {
        t = (ulong) N[k] * tL + c;
        w = (ulong) buf[j] - (t & 0xFFFF);
        t >>= 16;
        if (w < 0) {
            buf[j] = 0x10000 + w;
            t += 1;
        }
        else buf[j] = w;
        j++;
        w = (ulong) buf[j] - t;
        if (w < 0) {
            buf[j] = 0x10000 + w;
            c = 0x10000;
        }
        else {
            buf[j] = w;
            c = 0;
        }
    }
}

```

```

    if (tH > 0) {
        c = 0;
        j = e + 1;
        for (k = 1; k <= d; k++) {
            t = (ulong) N[k] * tH + c;
            w = (ulong) buf[j] - (t & 0xFFFF);
            t >>= 16;
            if (w < 0) {
                buf[j] = 0x10000 + w;
                t += 1;
            }
            else buf[j] = w;
        }
        if (k == d) break;
        j++;
        w = (ulong) buf[j] - t;
        if (w < 0) {
            buf[j] = 0x10000 + w;
            c = 0x10000;
        }
        else {
            buf[j] = w;
            c = 0;
        }
    }
    while (buf[n] == 0 && n > 0) n--;
    if (n == d && buf[d] < N[d]) break;
}

```

```

    for (k = 1; k <= n; k++) R[k] = buf[k];
    R[0] = n;
}

```

// Quick modulo A = A % m

ushort Modq(Int A, ushort m) { Modq

```

    register ulong z, t;
    ushort n, e;

    n = A[0];
    if (n == 0) return 0;

    for (e = 1; e <= n; e++) buf[e] = A[e];

    while (n > 1) {
        e = n - 1;
        t = ((ulong) buf[n] << 16) + buf[e];
        z = t/m;
        t -= z*m;
        buf[e] = t;
        if (t == 0) do e--;
        while (buf[e] == 0 && e > 0);
        n = e;
    }
    return buf[1] % m;
}

```

// Quick divide A = A / m

void Divq(Int A, ushort m) { Divq

```

    register ulong z, t;
    ushort n, e;

    n = A[0];
    if (n == 0) return;

    for (e = 1; e <= n; e++) {
        buf[e] = A[e];
        A[e] = 0;
    }

    while (n > 1) {
        e = n - 1;
        t = ((ulong) buf[n] << 16) + buf[e];
        z = t/m;

        if (z < 0x10000) A[e] = z;
        else {
            A[e] = z & 0xFFFF;
            A[n] = z >> 16;
        }
        t -= z*m;
    }

```



```

    buf[e] = t;
    if (t == 0) do e--;
    while (buf[e] == 0 && e > 0);
    n = e;
}
n = buf[1];
if (n >= m) A[1] = n/m;
if (A[A[0]] == 0) A[0] -= 1;
}

// Shift right X = X >> 1
void Shiftr(Int X) {
    register ulong t;
    register ushort i,j,c,d;

    d = X[0];
    if (d == 0) return;

    i = 0;
    j = 1;
    c = X[1] >> 1;
    while (j < d) {
        j++;
        t = (ulong) X[j] << 15;
        t += c;
        i++;
        X[i] = t & 0xFFFF;
        c = t >> 16;
    }
    if (c == 0) d -= 1;
    else X[d] = c;
    X[0] = d;
}

// Compare: {+1 0 -1} as {X > Y X == Y X < Y}
short Comp(Int X, Int Y) {
    register ushort d;

    d = X[0];
    if (d > Y[0]) return 1;
    if (d < Y[0]) return -1;

    while (X[d] == Y[d] && d > 0) d--;

    if (d == 0) return 0;
    if (X[d] > Y[d]) return 1;
    return -1;
}

// Convert from unsigned long to Int
void Set(Int X, ulong n) {
    if (n == 0) {
        X[0] = 0;
        return;
    }
    if (n < 0x10000) {
        X[0] = 1;
        X[1] = n;
        return;
    }
    X[0] = 2;
    X[1] = n & 0xffff;
    X[2] = n >> 16;
}

// Convert from Int to unsigned long
ulong Unset(Int X) {
    register ulong n;
    register ushort d;

    d = X[0];
    if (d == 0) return 0;
    n = (ulong) X[1];
    if (d == 1) return n;
    return (n + ((ulong) X[2] << 16));
}

// Number of Bits in X

```

```

ulong Bitsize(Int X) {
    register ushort d,t;
    register ulong n;

    d = X[0];
    if (d == 0) return 0;

    n = (ulong) d << 4;
    t = 0x8000;
    while ((t & X[d]) == 0) {
        n -= 1;
        t >>= 1;
    }
    return n;
}

// The greatest common divisor of A and N
ulong Gcd(Int A) {
    register long k;

    for (k = 0; k < 5; k++) buf[k] = N[k];

    while ((A[1] & 1) == 0) Shiftr(A);

    for (;;)
        switch (Comp(A,buf)) {
            case 1 : Dif(A,A,buf);
                    while ((A[1] & 1) == 0) Shiftr(A);
                    break;
            case -1 : Dif(buf,buf,A);
                    while ((buf[1] & 1) == 0) Shiftr(buf);
                    break;
            case 0 : return Unset(A);
        }
}

End of file Factor64.c

```

FACTOR64.H

// Header file for 64 bit factorization program.

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

```

```

#define ulong unsigned long
#define ushort unsigned short
typedef ushort * Int;

```

```

ulong inv(ulong,ushort);
short qrs(ushort,ushort);
ushort mrt(ushort,ushort);
ulong hrt(ulong,ushort);
ushort npr(ushort);

```

```

void Add(Int,Int,Int);
void Dif(Int,Int,Int);
void Mul(Int,Int,Int);
void Mulq(Int,ushort);
void Mod(Int);
ushort Modq(Int,ushort);
void Divq(Int,ushort);
void Shiftr(Int);
short Comp(Int,Int);
void Set(Int,ulong);
ulong Unset(Int);
ulong Bitsize(Int);
ulong Gcd(Int);

```

```

ushort buf[20], N[5];
ulong bufm[20];
double g;

```

// Odd primes below 0x800 (and their 10*logs)

```

ushort Prm[] = {
    3, 11, 5, 17, 7, 20, 11, 24, 13, 26,
    17, 29, 19, 30, 23, 32, 29, 34, 31, 35,
    37, 37, 41, 38, 43, 38, 47, 39, 53, 40,
    59, 41, 61, 42, 67, 43, 71, 43, 73, 43,
    79, 44, 83, 45, 89, 45, 97, 46, 101, 46,
    103, 46, 107, 46, 109, 46, 113, 47, 127, 48,
    131, 48, 137, 49, 139, 49, 149, 50, 151, 50,
    157, 50, 163, 50, 167, 51, 173, 51, 179, 51,

```

Bitsize

```

181, 51, 191, 52, 193, 52, 197, 52, 199, 52,
211, 53, 223, 54, 227, 54, 229, 54, 233, 54,
239, 54, 241, 54, 251, 55, 257, 55, 263, 55,
269, 55, 271, 56, 277, 56, 281, 56, 283, 56,
293, 56, 307, 57, 311, 57, 313, 57, 317, 57,
331, 58, 337, 58, 347, 58, 349, 58, 353, 58,
359, 58, 367, 59, 373, 59, 379, 59, 383, 59,
389, 59, 397, 59, 401, 59, 409, 60, 419, 60,
421, 60, 431, 60, 433, 60, 439, 60, 443, 60,
449, 61, 457, 61, 461, 61, 463, 61, 467, 61,
479, 61, 487, 61, 491, 61, 499, 62, 503, 62,
509, 62, 521, 62, 523, 62, 541, 62, 547, 63,
557, 63, 563, 63, 569, 63, 571, 63, 577, 63,
587, 63, 593, 63, 599, 63, 601, 63, 607, 64,
613, 64, 617, 64, 619, 64, 631, 64, 641, 64,
643, 64, 647, 64, 653, 64, 659, 64, 661, 64,
673, 65, 677, 65, 683, 65, 691, 65, 701, 65,
709, 65, 719, 65, 727, 65, 733, 65, 739, 66,
743, 66, 751, 66, 757, 66, 761, 66, 769, 66,
773, 66, 787, 66, 797, 66, 809, 66, 811, 66,
821, 67, 823, 67, 827, 67, 829, 67, 839, 67,
853, 67, 857, 67, 859, 67, 863, 67, 877, 67,
881, 67, 883, 67, 887, 67, 907, 68, 911, 68,
919, 68, 929, 68, 937, 68, 941, 68, 947, 68,
953, 68, 967, 68, 971, 68, 977, 68, 983, 68,
991, 68, 997, 69, 1009, 69, 1013, 69, 1019, 69,
1021, 69, 1031, 69, 1033, 69, 1039, 69, 1049, 69,
1051, 69, 1061, 69, 1063, 69, 1069, 69, 1087, 69,
1091, 69, 1093, 69, 1097, 70, 1103, 70, 1109, 70,
1117, 70, 1123, 70, 1129, 70, 1151, 70, 1153, 70,
1163, 70, 1171, 70, 1181, 70, 1187, 70, 1193, 70,
1201, 70, 1213, 71, 1217, 71, 1223, 71, 1229, 71,
1231, 71, 1237, 71, 1249, 71, 1259, 71, 1277, 71,
1279, 71, 1283, 71, 1289, 71, 1291, 71, 1297, 71,
1301, 71, 1303, 71, 1307, 71, 1319, 71, 1321, 71,
1327, 71, 1361, 72, 1367, 72, 1373, 72, 1381, 72,
1399, 72, 1409, 72, 1423, 72, 1427, 72, 1429, 72,
1433, 72, 1439, 72, 1447, 72, 1451, 72, 1453, 72,
1459, 72, 1471, 72, 1481, 73, 1483, 73, 1487, 73,
1489, 73, 1493, 73, 1499, 73, 1511, 73, 1523, 73,
1531, 73, 1543, 73, 1549, 73, 1553, 73, 1559, 73,
1567, 73, 1571, 73, 1579, 73, 1583, 73, 1597, 73,
1601, 73, 1607, 73, 1609, 73, 1613, 73, 1619, 73,
1621, 73, 1627, 73, 1637, 74, 1657, 74, 1663, 74,
1667, 74, 1669, 74, 1693, 74, 1697, 74, 1699, 74,
1709, 74, 1721, 74, 1723, 74, 1733, 74, 1741, 74,
1747, 74, 1753, 74, 1759, 74, 1777, 74, 1783, 74,
1787, 74, 1789, 74, 1801, 74, 1811, 75, 1823, 75,
1831, 75, 1847, 75, 1861, 75, 1867, 75, 1871, 75,
1873, 75, 1877, 75, 1879, 75, 1889, 75, 1901, 75,
1907, 75, 1913, 75, 1931, 75, 1933, 75, 1949, 75,
1951, 75, 1973, 75, 1979, 75, 1987, 75, 1993, 75,
1997, 75, 1999, 76, 2003, 76, 2011, 76, 2017, 76,
2027, 76, 2029, 76, 2039, 76, 0, 0 };

```

End of file Factor64.h



**To receive
information
on any products
advertised
in this issue,
send your request
via Internet:
productinfo@xplain.com**



By Randy Thelen, Apple Computer, Inc.

Threading Your Apps

Tying it all together

The Thread Manager is a system software extension which allows applications to have multiple threads of execution. With multiple threads of execution you can easily move the processing of relatively lengthy operations into the background thus creating a more responsive application for your users. In this article we'll learn what the terminology is, we'll explore the programming model you'll want to employ to make best use of threads, and we'll examine the application programming interface (API).

The Threads Manager extension version 2.0.1 (68K and PowerPC based threads) is available for distribution with your application from Apple for \$50 (through APDA) and it is built in to System 7.5. Further, Apple's future O/S endeavors will be threaded. If you employ a threaded model in current application structure, it will carry over transparently (or requiring only minor API changes) to the next generation system software.

TERMINOLOGY

A few words from the man. Without using any names (Eric

Anderson), there's this Smart Guy™ over here at Apple who was instrumental in actually packaging the Threads Manager for shipment. He wrote the following paragraphs for developers (I felt it fitting to enclose them in this article for you):

"The Thread Manager is the current MacOS solution for lightweight concurrent processing. Multithreading allows an application process to be broken into simple subprocesses that proceed concurrently in the same overall application context. Conceptually, a thread is the smallest amount of processor context state necessary to encapsulate a computation. Practically speaking, a thread consists of a register set, a program counter, and a stack. Threads have a fast context switch time due to their minimal context state requirement and operate within the application context which gives threads full application global access. Since threads are hosted by an application, threads within a given application share the address space, file access paths and other system resources associated with that application. This high degree of data sharing enables threads to be 'lightweight' and the context switches to be very fast relative to the heavyweight context switches between Process Manager processes.

"An execution context requires processor time to get anything done, and there can be only one thread at a time using the processor. So, just like applications, threads are scheduled to share the CPU, and the CPU time is scheduled in one of two ways. The Thread Manager provides both cooperative and preemptive threads. Cooperative threads explicitly indicate when they are giving up the CPU. Preemptive threads can be interrupted and gain control at (most) any time. The basis for the difference is that there are many parts of the MacOS and Toolbox that can not function properly when interrupted and/or executed at arbitrary times. Due to this restriction, threads using such services must be cooperative. Threads that do not use the Toolbox or OS may be preemptive.

"Cooperative threads operate under a scheduling model

Randy Thelen – Randy (sometimes known as Random) is the kind of Apple engineer who just keeps coming up with wacky ideas that just might work. In his spare time, he's been playing with Threads way too much, and was last seen listening to They Might Be Giants while excitedly showing off fast, color, bit-shuffling code.

A DEVELOPER'S BEST FRIEND

Don't Bury Your Profits!

Software producers will lose over 7 billion dollars worldwide to software piracy this year. This number would be even higher if it weren't for PACE Anti-Piracy. For nearly a decade, PACE has provided Macintosh software producers with the most effective means of protecting against piracy.

Your Sales Will Increase But Your Tech Support Calls Won't.

Not just copy protection, **MacEncrypt™** is a complete key diskette and verbal authorization system. With **MacEncrypt** you can protect your profits while providing your customers with the flexibility and ease of use they require. Unlike other products on the market, especially hardware keys and decoder rings, your customers will have a positive experience with your **MacEncrypt** protected software.

Features:

- Protects all disk formats, including CD ROM's. Requires no special media.
- Authorize your customers over the phone.
- Protected software can be installed to hard disks, compatible with disk optimizers.
- Protected diskettes can't be copied by disk copy programs.
- Turn-key installation for applications; you don't have to be a programmer to protect your app.
- The protection can be custom installed to any software—not just apps.
- Complete integration with the Apple Installer and StuffIt InstallerMaker™.
- Automatically protects against virus infection.

For a protected sample diskette and more information about the MacEncrypt system, call or write **PACE Anti-Piracy** 1082 Glen Echo Ave, San Jose, CA 95125
(408) 297-7444 • Fax (408) 297-7441 • AppleLink PACE.AP



P A C E
ANTI-PIRACY

"Copy protection that isn't superb costs a company more than it saves. PACE products offer unparalleled security and compatibility at a great price. PACE is a rare company dedicated to making the unthinkable possible: copy protection that's hassle-free, for both my end-users and my support staff."

Stephen Greenfield
President
Screenplay Systems, Inc.

"PACE Anti-Piracy software is a reliable means of limiting illegal distribution of our software and enforcing our licensing agreements."

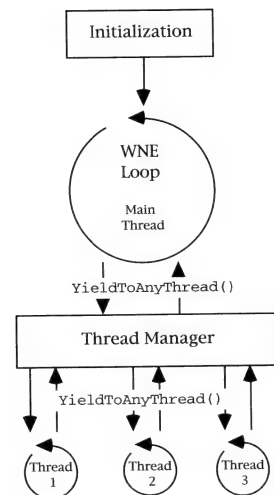
Mike Aaron
Opcode Systems

similar to the Process Manager, wherein they must make explicit calls for other cooperative threads to get control. As a result, they are not limited in the calls they can make as long as yielding calls are properly placed. Preemptive threads operate under a time slice scheduling model; no special calls are required to surrender the CPU for other preemptive or cooperative threads to gain control. For threads which are compute-bound or use MacOS and Toolbox calls that can be interrupted, preemptive threads may be the best choice; the resulting code is cleaner than if partial results were saved and control then handed off to other threads of control."

That said, let's make it clear early on that preemptive threads are not supported on the PowerMacintosh at this time and therefore developers are strongly encouraged to use cooperative threads. (In reality, this hasn't posed much of a problem for most developers, given the number of restrictions for preemptive threads.)

PROGRAMMING MODEL

In this section, we'll discuss how to structure your program. Here's a block diagram of a basic application:



WNE Loop is, of course, that block of code which you cycle through more rapidly than `GetCaretTime()` ticks expire. Right? If not, that's one of the first things we'll learn about the Threads programming model. It's actually possible to cycle through your event loop more quickly, giving your customer a more responsive computer.

A *thread* is, of course, the center piece of this article.

New BBEdit 3.0 High Performance Text Editing

"It still doesn't suck."

—Tom Emerson, Software Engineer, Symantec Corporation

BBEdit on a PowerMac gives you even faster text editing, searching and transformation.

More than just a pretty face, BBEdit 3.0 kicks ass.

Among other things, it offers:

- Scripting from AppleScript, Frontier 3.0, or any other OSA-compatible scripting system.
- Macintosh Drag and Drop to speed text editing.
- Integrated PopupFuncs™ technology for easy navigation of source files.
- Integrated support for Symantec C++, and THINK Reference, and MPW ToolServer.
- The ability to compare projects or folders, as well as files.
- A non-stick coating for easy clean up.

...and the list goes on...

Now we're screaming.

BBEdit edits, searches, transforms, compares and massages text between two and ten times faster on a Power Mac 8100 than on a Quadra 610.

"...most people will find that BBEdit offers everything they need and is easy to use."

—MacTech, June 1994

Tell me more! Tell me more!

Check the Mail Order Store section at the back of this issue for info about ordering BBEdit.

For a free demo disk, send us e-mail:

Internet: bbedit@world.std.com

AppleLink: BARE.BONES CompuServe: 73051,3255



We also have "It doesn't suck." t-shirts that don't either. Call us to order. © 1994 Bare Bones Software, 1 Larkspur Way #4, Natick, MA 01760. Phone: 508/651-3561. Fax: 508/651-7584. By the way: BBEdit is a trademark of Bare Bones Software, Inc. All other trademarks are properties of their respective holders.

Yielding is the process of giving up the CPU for some period of time. As Eric mentioned in his paragraphs was that preemptive threads are yielded inherently by a timer interrupt; they do not yield. Cooperative threads yield. They call `YieldToAnyThread()`. When we examine the API, we'll see this call. For now, let's just remember that yielding is the process of asking the Thread Manager to find the thread of execution which should execute next.

The *circular shapes* represent the looping nature of a thread. As it turns out, not all threads loop. Some follow some series of steps: A, B, C, ..., etc. Those threads, if they take a long time, should be making asynchronous I/O calls with yield calls where `SyncWait` would normally execute. For example,

```
AsyncFileManagerCall( &pb);
while( pb.ioResult > 0)
YieldToAnyThread();
```

In our block diagram, we find the two steps most programs have: the initialization and the `WaitNextEvent` loop. The WNE loop looks something like this (code snippet from Sprocket, courtesy of Dave Falkenburg – thanks Dave, let's do lunch real soon):

```
Boolean    gDone = false;
Boolean    gMenuBarNeedsUpdate = false;
long       gRunQuantum = GetCaretTime();
long       gSleepQuantum = 3;
RgnHandle  gMouseRegion = nil;
```

```
Boolean    gHasThreadManager = false;

void MainEventLoop(void)
{
    EventRecord  anEvent;
    unsigned long nextTimeToCheckForEvents = 0;

    while (!gDone)
    {
        if (gHasThreadManager)
            YieldToAnyThread();

        if (gMenuBarNeedsUpdate)
        {
            gMenuBarNeedsUpdate = false;
            DrawMenuBar();
        }

        if ((gRunQuantum == 0) ||
            (TickCount() > nextTimeToCheckForEvents))
        {
            nextTimeToCheckForEvents = TickCount() + gRunQuantum;
            (void) WaitNextEvent( everyEvent, &anEvent,
                                gSleepQuantum, gMouseRegion);
            HandleEvent(&anEvent);
        }
    }
}
```

Immediately we find two things: PowerMacintosh `WaitNextEvent` "smarts" and support for the Thread Manager. The WNE "smarts" is simply a mechanism for throttling the frequency with which `WaitNextEvent` is called on a PowerMacintosh. (The issue here, if you're not already familiar

NEW

absoft
development tools and languages

NEW

C++

for Power Macintosh

- Native Power Macintosh C++ compiler
- ANSI and K&R C
- 100% Plum Hall Validated
- Templates
- High Speed Math Library

- Fx™ multi-language debugger
- Power-Link — Absoft's native linker
- Apple's MPW development environment
- Link compatible with Absoft's F77 SDK
- Open, flexible development environment

Available Now: \$399.00 Complete

Tel: (810) 853-0050 • Fax : (810) 853-0108 • AppleLink: absoft • Internet: c@absoft.com

with it, is that WNE invokes a context switch from PowerPC code to 68K emulation and if the application calls WNE too frequently then performance goes into the proverbial toilet.)

The Thread Manager support is, as you can see, petty. (There was, one would hope, the appropriate check for the presence of the Thread Manager. We'll see that code in a couple pages.)

The Toolbox trap YieldToAnyThread() uses trap number \$ABF2 (which goes by the name ThreadDispatch, and gets a selector in D0). If we glance back at our block diagram, we see that the Toolbox trap YieldToAnyThread() calls into the Thread Manager. It yields the CPU to one of the other threads of execution within the program context. Each thread is then responsible for calling YieldToAnyThread() frequently enough that two things will happen: one, the cursor, if you've got one, will blink with reasonable consistency; second, you'll want events (mouse downs, etc.) to be handled pretty quickly.

With regard to the insertion point blinking, the rule of thumb here really varies: if you've got an arbitrary number of threads (potentially greater than GetCaretTime()'s smallest value), you'll want to yield before a tick expires; if you've only got a few threads, you may want to time your processing using the same kind of TickCount() > someQuantum algorithm as what Dave's done above.

Regarding events, there is something very important you should be familiar with. The Thread Manager will check to see

if any events are pending for the application each time a thread yields. If there are events (or there is an event), the thread that gets time next is the main thread. If the main thread then yields without processing the event (or all events), another thread is executed, but upon the next yield, the main thread will then get time again. It's an algorithm that ensures two things: first, the main thread gets time often enough to handle incoming events as quickly as threads yield, and second, events are not "starved" (threads are always hungry for CPU time).

Obviously these over-simplified rules won't work for everybody. Hopefully, after you've read the bulk of these articles, you'll get some feel for where to begin your experimentation for coming up with a processing time model that works best for your application and thread requirements.

Your application will create these threads whenever it's appropriate to do so. In SortPics (one of the apps included on the source disk and online sites this month), for example, a thread is created when a window is opened. In ThreadBrot (a threaded Mandelbrot, also on disk and online), threads are created as rectangles within the complex number plane are halved (it's a divide-and-conquer algorithm) until some lower bound rectangle size is reached — recursive algorithms must always have a base case. In Steve Sisak's article next month, we'll see threads created for sending AppleEvents — who'd a thunk it? In short, you're free to create threads during the



SOFTWARE FOR SALE? NO ADVERTISING BUDGET?

Try listing your product in MacTech Magazine's Mail Order Store.
Classified advertising at a cost effective rate.

For more information, call:

Voice: 310/575-4343 • Fax: 310/575-0925

AppleLink, GENIE & America Online: MACTECHMAG

CompuServe: 71333,1064 • Internet: marketing@xplain.com

MacTechMAGAZINE™

FOR MACINTOSH PROGRAMMERS & DEVELOPERS

execution of your program whenever your process is given time by the process manager. In fact, you can preallocate threads into a pool (which the Thread Manager will maintain for you) and then you can spawn new threads of execution during preemptive threads or during interrupts (remember, in 68K land your A5 world must be set correctly; for PowerPC applications, you must make this call from PowerPC code — or the Threads Manager will crash your application).

API

There are only a couple of calls you need to know about: `NewThread`, `YieldToAnyThread`. With these two calls, you could make your programs amazingly responsive. With other calls we'll discuss, you can do much more.

```
FUNCTION NewThread (threadStyle: ThreadStyle;  
    threadEntry: ThreadEntryProcPtr;  
    threadParam: LONGINT;  
    stackSize: Size;  
    options: ThreadOptions;  
    threadResult: LongIntPtr;  
    VAR threadMade: ThreadID):OSErr;
```

You'll call `NewThread` whenever you wish to create a new thread of execution. `threadStyles` are `kPreemptiveThread` and `kCooperativeThread`. Again, unless you're writing a program for use only on a 68K machine, you'll want to limit yourself to `kCooperativeThread`.

Your `threadEntry` is the address of the function which will get executed when the thread is first executed. You should remember that this entry point will be called only *once* for your thread. From then on, when you call `YieldToAnyThread`, your thread will continue execution from the instruction immediately following the yield call.

The `threadParam` is passed to your thread when it is first called. This allows you to pass a value or the address of an arbitrarily-large data structure to your thread.

The `stackSize` field defines the size that you believe to be adequate to maintain context switch information, satisfy Toolbox stack requirements, and fulfill your thread's stack needs. If you pass zero for this field, you will get the default stack size. (You can inspect this size by calling `GetDefaultThreadStackSize`, and you can set it by calling `SetDefaultThreadStackSize`.)

The `options` field allows you to define some characteristics about the thread you want created: `kNewSuspend` creates a thread which is not inherently eligible for execution; `kFPUNotNeeded` denotes that (on 68K machines) the FPU context will not be saved on the stack during context switches (this option has no effect on the PowerMacintosh implementation of threads).

The `threadResult` field will be filled in when your thread actually terminates. Its return value is placed in the

Choosing an installer program should be easy

- Easy for your users** DragInstall's unique drag-and-drop interface makes installing your software a piece of cake for your users.
- Easy for you** DragInstall's Builder utility allows you to create complex installers in minutes not days. And without the need to learn a complicated scripting language.
- Easy on your wallet** DragInstall's one-time fee allows you to distribute an unlimited number of installers. No yearly renewals, no royalties, no hassle!
- Easy as DragInstall** See for yourself just how easy DragInstall is. Contact us for a free demo disk and information kit

New in DragInstall 1.5.3

- New compression algorithm saves 15-20% more space than previous versions.
- International templates allow building of French, German, and Italian installers.
- New external allows creation of aliases for installed files.

Ray Sauers Associates, Inc.

1187 Main Avenue, Suite 1B • Clifton, NJ 07011-2252 • Voice: 201-478-1970 • Fax: 201-478-1513 • AppleLink: D1922 • AOL: SAUERS

memory pointed to here. The NIL case is handled correctly (that is, nothing is put there if you pass in NIL; this avoids a write to memory location 0).

The `threadMade` is the `ThreadID` of the thread just created. You'll use this ID to refer to threads in the future. For example, if you wish to kill a thread, you may call `DisposeThread(threadID);`.

And last, but not least, there are error codes to be interpreted. Obviously `noErr` is a good thing. `memFullErr` means that the thread wasn't created because there wasn't room for the stack or thread structure. `paramErr` is returned if you attempt to create a `kPreemptive` thread on a PowerMacintosh or if you don't use one of the two defined values for thread type in the thread type field.

OK. So now you've got a dozen threads running rampant in your application and you need to know how to switch from one to the other to the other to the other and back to the main thread of execution (your WNE loop) so that you can actually get some processing done. No problemo, señor y señorita programmer.

`YieldToAnyThread()` will get you around your threads quite simply. It takes no parameters and will simply invoke the Thread Manager scheduler to determine which thread should be next to execute. There will be occasions when you will know best which thread should execute next. On these occasions, you will want to call `YieldToThread(threadID)`.

There are several other useful threads calls, and I'll cover a few here. For more detail, the entire Thread Manager specifications follow right after this article.

PROGRAMMING EXAMPLES

Let's look at some code examples to see how these routines are

used by a real program. The code we'll look at does three things: 1) Checks for the complete presence of the Thread Manager; 2) Create the thread; 3) `YieldToAnyThread()`.

Checks for the complete presence of the Thread Manager

Checking for the complete presence of the Thread Manager is pretty simple. First, you need to call `Gestalt` (thanks to the magic of glue, even `Gestalt` is compatible across all versions of system software back to 4.3) and check that the 'thds' selector is present. Second, you'll need to check to see that the `gestaltThreadMgrPresent` bit is set. No problem.

For PowerPC code, you need two additional tests. Third, you need to see that the native library is present (this was added to threads with ThreadManager 2.0 -- thanks to Brad Post). Fourth, you need to confirm that the Code Fragment Manager (CFM) actually resolved the Thread Manager code fragment with your application (a shared library).

The reason for the fourth test is because of two conditions: A) the `ThreadsLib` library may not have loaded -- low memory conditions with VM off, for example; and, B) you should be "weak" linking your application with the `ThreadsLib` library. This way, if the `ThreadsLib` library isn't present on PowerPC machines, your native code can handle the conditional appropriately (a modeless dialog might mention that some features won't function because some system software features weren't found) as opposed to the Finder bringing up a modal dialog, "ThreadsLib couldn't be found." Like, what does that mean to most of your users?

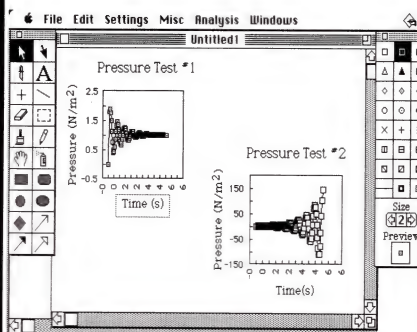
Without further ado, here's the code:

```
// Test for the presence of the threads manager
// 1. Is the gestalt selector defined? If not, we bail immediately
// 2. If we're compiling PPC native code, then check for the ThreadsLibrary
```


New Release

- 19 plot types
- Data analysis
- Interactive plot editing
- Batch processing options
- Multiple plot windows
- Multiple plots in same window
- Sub-scripting and super-scripting
- Number points limited by memory
- Update plot from calling application
- Display values in spreadsheet
- Templates to initialize plot characteristics
- Set plot characteristics from application

SuperPlot PRO™



Generate presentation quality plots with simple subroutine/procedure calls from your Fortran, Pascal, or C program

For free demo and info contact

SuperSoft

498 E. Robin Rd.
Orem, UT 84057
(801) 225-4356
Fax: (801) 226-6276
Applelink: SuperSoft.UT

```
// 3. Also, if we're native, check that CFM actually linked my app to the library
// 4. Is the Thread Mgr Present bit set to True? If not, bail
```

```
if( Gestalt( gestaltThreadMgrAttr, &threadGestaltInfo) != noErr ||
    #if defined(powerc) || defined (__powerc)
        threadGestaltInfo & (1<<gestaltThreadsLibraryPresent) == 0 ||
        (Ptr) NewThread == kUnresolvedSymbolAddress ||
    #endif
    threadGestaltInfo & (1<<gestaltThreadMgrPresent) == 0)
{
    // This is the bail clause. Yours may be more elaborate
    printf( "Threads Mgr isn't present... Can't run the test.\n");
    return;
}
```

Create the Thread

There's really no mystery about creating new threads, but we'll walk through an example, just the same.

```
pascal void *MyThread_A( void *refCon);
```

```
myErr = NewThread( kCooperativeThread, MyThreadProc,
    (void *)0, 0, kFPUNotNeeded + kCreateIfNeeded,
    (void**)nil, &threadID);
```

The NewThread parameters are: thread type, entry procedure, refcon, size of stack, thread options, where to put a return value (when the thread entry procedure "returns"), and, last, a place to put the ID which identifies the thread just created.

So, what this call does is create a cooperative thread, executing the function MyThreadProc with 0 for a refcon with the default stack size. For options, the floating point unit is not needed and the ThreadManager is given the OK to allocate the thread memory if it is needed (remember the thread pool concept from earlier? yea, that's what this option affects). If the thread has a return value, I don't care about it (by passing nil as a Ptr to a void *). Last, when the thread is created I want the thread ID stored in the variable called threadID.

A note to C++ users: Some implementations of C++ (THINK C 5.0 with Objects, for example) allow the creation of

Handle-based objects. If your object is Handle-based and you pass the address of an Object variable to this function (or any Mac Toolbox function which may move memory during its operation), your object may move! This is a bad thing because the ptr to your object variable will no longer point to your object variable after your object data block moves. Solutions to this conundrum are well understood: the best is to pass the address of a stack-based variable (i.e., local variable). The alternative (which is much uglier, but also viable) is to lock your object with HLock. I suggest the local variable.

After the thread is created, you can call YieldToAnyThread and the first line of the thread entry procedure will execute. A word regarding thread execution order: if you have more than one thread, there is no guarantee that the threads will execute in any particular order from yield call to yield call. The only exception is that the main thread (the thread that is created automatically which contains your main() function) will execute if an event is pending in the event queue. This feature means that if the user presses the mouse button, just as fast as your thread yields, the main thread will be executed and it will be free to call WaitNextEvent, which will simply return with a valid (mdown) event and then you can process the event accordingly.

YieldToAnyThread()

At the heart and soul of a thread, there is a thread procedure which will be executed: once. When that function completes, the thread is deemed dead and is no longer eligible for execution. So for example, logic which reads:

```
main()
{
    NewThread( MyThreadProc);
    while( true)
```



```

    YieldToAnyThread();
}

pascal void *MyThreadProc( void *refcon)
{
    printf( "Hello from thread\n");
    return nil;
}

```

will print only one line, "Hello from thread." Therefore, once your thread gets control, it needs to have its own looping logic built in. Like,

```

pascal void *MyThreadProc( void *refcon)
{
    for( i = 0; i < 5; i++)
    {
        printf( "Hello from thread\n");
        YieldToAnyThread();
    }
    return nil;
}

```

This will print 5 "Hello from thread" lines before it terminates.

Please look at the source code which accompanies this article. Next month, look for a case study on an app I wrote called SortPicts (and maybe another about AppleEvents and Threads). Read the Thread Manager Documentation. Reread Hitchhiker's Guide to the Galaxy. Then take some time off. Enjoy life more. Read back issues of MacTutor and MacTech.

Here's the code to TestThreads.c. It's followed by the output it generates.

TESTTHREADS.C

```

#include <threads.h>
#include <stdio.h>
#include <GestaltEqu.h>
#ifdef __powerc
#include <FragLoad.h>
#endif

pascal void *MyThread_A( void *refCon);
pascal void *MyThread_B( void *refCon);
pascal void *MyThread_C( void *refCon);

void main( void)
{
    OSErr    errWhatErr;
    int      i;
    ThreadID threadID_A, threadID_B, threadID_C;
    long     threadGestaltInfo;

    // Test for the presence of the threads manager
    // 1. Is the gestalt selector defined? If not, we bail immediately
    // 2. If we're compiling PPC native code, then check for the ThreadsLibrary
    // 3. Also, if we're native, check that CFM actually linked my app to the library
    // 4. Is the Thread Mgr Present bit set to True? If not, bail

    if( Gestalt( gestaltThreadMgrAttr, &threadGestaltInfo) != noErr ||
#ifdef __powerc
        || defined( __powerc )
        threadGestaltInfo & (1<<gestaltThreadsLibraryPresent) == 0 ||
        (Ptr) NewThread == kUnresolvedSymbolAddress ||
#endif
        threadGestaltInfo & (1<<gestaltThreadMgrPresent) == 0)
    {
        // This is the bail clause. Yours may be more elaborate
        printf( "Threads Mgr isn't present... Can't run the test.\n");
        return;
    }

    // Create 3 threads: A, B, C
    errWhatErr = NewThread( kCooperativeThread, MyThread_A,
        (void *)0, 0, kFPUNotNeeded + kCreateIfNeeded,
        (void**)nil, &threadID_A);
}

```

```

errWhatErr = NewThread( kCooperativeThread, MyThread_B,
    (void *)0, 0, kFPUNotNeeded + kCreateIfNeeded,
    (void**)nil, &threadID_B);
errWhatErr = NewThread( kCooperativeThread, MyThread_C,
    (void *)0, 0, kFPUNotNeeded + kCreateIfNeeded,
    (void**)nil, &threadID_C);

// Simple loop to test Yielding
for( i = 0; i < 5; i++) {
    printf( "This is thread main\n");
    YieldToAnyThread();
}

pascal void *MyThread_A( void * /* refCon */)
{
    int      i;
    for( i = 0; i < 5; i++) {
        printf( "----- A ----- \n");
        YieldToAnyThread();
    }
    return nil;
}

pascal void *MyThread_B( void * /* refCon */)
{
    int      i;
    for( i = 0; i < 5; i++) {
        printf( "----- B ----- \n");
        YieldToAnyThread();
    }
    return nil;
}

pascal void *MyThread_C( void * /* refCon */)
{
    int      i;
    for( i = 0; i < 5; i++) {
        printf( "----- C ----- \n");
        YieldToAnyThread();
    }
    return nil;
}

```

TestThreads produces the following output when run (note that, although the order of execution appears deterministic, you shouldn't rely on that behavior):

This is thread main

```

----- A -----
----- B -----
----- C -----

```

This is thread main

```

----- A -----
----- B -----
----- C -----

```

This is thread main

```

----- A -----
----- B -----
----- C -----

```

This is thread main

```

----- A -----
----- B -----
----- C -----

```

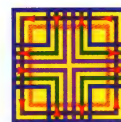
This is thread main

```

----- A -----
----- B -----
----- C -----

```





By Apple Computer, Inc.

Thread Manager for Macintosh Applications

Apple's Development Guide

This article will provide the reader with the motivation, architecture, and programmatic interface of the Thread Manager. The architecture section will give some detail of how the Thread Manager is integrated into the Macintosh environment and some of the assumptions made in its design. The programmatic interface will then be described with commentary on the use of each of the routines. The end contains information such as current known bugs and compatibility issues.

Product Definition

The Thread Manager is the current MacOS solution for lightweight concurrent processing. Multithreading allows an application process to be broken into simple subprocesses that proceed concurrently in the same overall application context. Conceptually, a thread is the smallest amount of processor context state necessary to encapsulate a computation. Practically speaking, a thread consists of a register set, a program counter, and a stack. Threads have a fast context switch time due to their minimal context state requirement and operate within the application context which gives threads full application global access. Since threads are hosted by an application, threads within a given application share

the address space, file access paths and other system resources associated with that application. This high degree of data sharing enables threads to be "lightweight" and the context switches to be very fast relative to the heavyweight context switches between Process Manager processes.

An execution context requires processor time to get anything done, and there can be only one thread at a time using the processor. So, just like applications, threads are scheduled to share the CPU, and the CPU time is scheduled in one of two ways. The Thread Manager will provide both cooperative and preemptive threads. Cooperative threads explicitly indicate when they are giving up the CPU. Preemptive threads can be interrupted and gain control at (most) any time. The basis for the difference is that there are many parts of the MacOS and Toolbox that can not function properly when interrupted and/or executed at arbitrary times. Due to this restriction, threads using such services must be cooperative. Threads that do not use the Toolbox or OS may be preemptive.

Cooperative threads operate under a scheduling model similar to the Process Manager, wherein they must make explicit calls for other cooperative threads to get control. As a result, they are not limited in the calls they can make as long as yielding calls are properly placed. Preemptive threads operate under a time slice scheduling model; no special calls are required to surrender the CPU for other preemptive or cooperative threads to gain control. For threads which are compute-bound or use MacOS and Toolbox calls that can be interrupted, preemptive threads may be the best choice; the resulting code is cleaner than if partial results were saved and control then handed off to other threads of control.

PART 1: REQUIREMENTS SUMMARY

The Thread Manager is an operating system enhancement that will allow applications to make use of both cooperative & preemptive multitasking within their application context on all 680x0 based Macintosh platforms, and cooperative multitasking on PowerPC based Macintoshes. There are two basic types of threads (execution contexts) available: cooperative and

This is Apple's Thread Manager documentation, reprinted with permission. Copyright © 1994, Apple Computer, Inc.

preemptive. The different types of threads are distinguished by their scheduling models.

The benefits of per-application multitasking are numerous. Many applications are best structured as several independent execution contexts. An image processing application might want to run a filter on a selected area and still allow the user to continue work on another portion of an image. A database application may allow a user to do a search while concurrently adding entries over a network. With the Thread Manager, it is now possible to always make applications responsive to the user, even while executing other operations. The Thread Manager also gives applications an easy way to organize multiple instances of same or similar code. In each example it is possible to write the software as one thread of execution, however, application code may be simplified by writing each class of operation as a separate thread and letting the Thread Manager handle the interleaving of the threaded execution contexts.

These examples are not intended to be exhaustive, but they indicate the opportunities to exploit the Macintosh system and build complex applications with this technology. The examples show that the model for multiple threads of control must support a variety of applications and user environments. The Thread Manager architecture will, where possible, use the current Macintosh programming paradigms and preserve software compatibility. The Thread Manager enhances the programming model of the Macintosh for there is little need to develop Time Manager or VBL routines to provide the application with a preemptive execution context. There is also no need to save the complete state of a complex calculation in order to make WaitNextEvent or GetNextEvent calls to be user responsive - simply yield to give the main application thread a chance to handle interface needs.

Hardware Compatibility

The 680x0 version of the Thread Manager has the same hardware requirements as System 7.0, that is, at least 2 megabytes of memory, and a Macintosh Plus or newer CPU. The power version of the Thread Manager requires any Power Macintosh.

Software Compatibility

System 7.0 or greater is required for the 680x0 version of the Thread Manager to operate. The power version of the Thread Manager requires system software for Power Macintosh platforms.

Existing applications that know nothing about the Thread Manager have nothing to fear. The extent of the Thread Manager's influence is to set up the main application thread when the application is launched, and to make an appearance every so often as the preemption timer fires off. Because there is only the application thread, the preemption timer has nothing to do and quietly returns. Thus, the Thread Manager is nearly transparent to existing applications, and no compatibility concerns are expected. New applications, of course, can reap

the full benefits of concurrent programming, including a fairly powerful form of multitasking.

The power version of the Thread Manager is built as a Shared Library, named ThreadsLib, that is fully integrated into the Thread Manager.

Intended Users

Developers will gain the ability to have multiple, concurrent, logically separate threads of execution within their application. The Thread Manager will provide Macintosh developers with a state of the art foundation for building the next generation of applications using a multi-threaded application programming environment. Another, less obvious user is system software which operates in the application context. The rule of thumb is: Code which operates only within an application context can use the Thread Manager, code that does not, can not.

Programmatic Interface Description

The Thread Manager performs creation, scheduling and deletion of threads. It allows multiple independent threads of execution within an application, each having its own stack and state. The client application can change the scheduler or context switch parameters to optimize an application for a particular usage pattern.

Applications will interface with the Thread Manager through the use of the Trap mechanism we know and love. The API is well defined, compelling, and easy to use—no muss, no fuss. For those who need to get down and dirty, the Thread Manager provides routines to modify the behavior of the scheduling mechanism and context switching code.

The API goes through a single trap: ThreadDispatch. Parameters are transferred on the stack and all routines return an OSErr as their result. The trap dispatch numbers have both a parameter size and a routine number encoded in them which allows older versions of the Thread Manager to safely reject calls implemented only by newer versions. A paramErr is returned for calls not implemented.

The ThreadsLib is a shared library so there is no performance hit due to a trap call, when using the Thread Manager API. There is a distinction between 680x0 threads and power threads. A 680x0 application may only use 680x0 threads, and power applications may only use power threads. Mixing thread types (power or 680x0) within an application type (power or 680x0) is considered a programming error and is not supported.

Performance

The context switch time for an individual thread is negligible due to the minimal context required for a switch. The default context saved by the Thread Manager includes all processor data, address, and FPU (when required) registers. The thread context may be enhanced by the application (to include application specific context) which will increase context switch times (your mileage may vary).

Both cooperative and preemptive threads are eligible for

execution only when the application is switched in by the process manager. In this way, all threads have the full application context available to them and are executed only when the application gets time to run.

The interleave design of one cooperative context between every preemptive context guarantees that threads which can use the Toolbox (cooperative threads) will be given CPU time to enhance user interface performance.

PART 2: FUNCTIONAL SPECIFICATIONS

Features Overview

Per-application thread initialization is completed prior to the entering the application, which allows applications to begin using the Thread Manager functions as soon as their main routine begins execution. Thread clean up is not required as this is done by the Thread Manager at application termination time.

Applications are provided with general purpose routines for thread pool creation, counting, allocation and deletion. Basic scheduling routines are provided to acquire the ID of the currently executing thread and to yield to any thread. Preemptive thread scheduling routines allow a thread to disable preemption during critical sections of code. Advanced scheduling routines give the ability to yield to a particular thread, and get & set the state of any thread. Mechanisms are also provided to customize the thread scheduler and add custom context switching routines.

Software Design & Technical Description

Installation: During system startup, the Thread Manager is installed into the system and sets up system-wide globals and patch code.

Initialization: Per-application initialization is done prior to entering the application. This allows applications to take advantage of the Thread Manager functions as soon as they begin execution of the main application thread. Important: The Memory Manager routine MaxApplZone must be called before any thread other than the main application thread allocates memory, or causes memory to be allocated (see the Constraints, Gotchas & Bugs section for more information).

Cleanup: The Thread Manager is called by the Process Manager when an application terminates. This gives the Thread Manager a chance to tear down the threading mechanism for the application and return appropriate system resources, such as memory.

Control: The Thread Manager gets control in three ways. The straightforward way is through API calls made by a threaded application. All calls to the Thread Manager are made through the trap ThreadDispatch (0xABF2). The less straightforward way is via hardware interrupts to give the Thread Manager preemption scheduler a chance to reschedule preemptive threads. For power applications, the Thread Manager is called through the use of library calls to the Thread Manager shared library.

Thread Types: The Thread Manager allows applications to

create and begin execution of two types of threads: cooperative and preemptive. Cooperative threads make use of a cooperative scheduling model and can make use of all Toolbox and operating system functions. This type of thread has all the rights and privileges of regular application code, which includes the use all Toolbox and OS features available to applications today. For 680x0 applications only, preemptive threads make use of a preemptive scheduling model and may not make use of Toolbox or operating system services; only those Toolbox or operating system services which may be called from an interrupt service routine may be called by preemptive threads. The Toolbox and OS calling restrictions include traps like LoadSeg which get called on behalf of your application when an unloaded code segment needs to be loaded. **Important:** Be sure to preload all code segments that get used by preemptive threads. Also note that preemptive threads, like interrupt service routines, may not make synchronous I/O requests.

Main Application Thread: The main application thread is a cooperative thread and contains the main entry point into the application. This thread is guaranteed to exist and can not be disposed of. All applications will have one main application thread, even if they are not aware of the Thread Manager. The main application thread is defined to be responsible for event gathering (via WaitNextEvent or GetNextEvent). If events are pending in the application event queue when a generic yield call is made (no thread ID is specified) by another cooperative thread, the Thread Manager scheduler chooses the main application thread as the next cooperative thread to run. This gives the main application thread a chance to handle events for user responsiveness.

Memory Management: The Thread Manager provides a method of creating a pool of threads. This allows the application to create a thread pool early in its execution before memory has been used or overly fragmented. Threads may be removed from the thread pool on a stack size best fit or exact match basis for better thread pool management. Thread data structures can be allocated at most any time, provided the Memory Manager routine MaxApplZone has been called (see the Constraints, Gotchas & Bugs section for more information).

Important: It is considered a programming error to allocate memory, or cause memory to be allocated, during preemptive execution time or from any thread other than the main application thread before MaxApplZone has been called.

Thread stack requirements are determined by the type of thread being created and the application's specific use of that thread. The stack size of a thread is entirely up to the developer - the Thread Manager can only let the developer know the default size and the currently available thread stack space. Cooperative threads may make Toolbox and OS calls which require a larger stack than threads which can not make such calls. Stack based parameter passing from a thread is fully supported since the Thread Manager does not BlockMove thread stacks in and out of the application's main stack area. Each thread has its own stack which does not move once allocated.

Scheduling: All scheduling occurs in the context of the currently executing application. When the application gets time to run via the Process Manager, the application's threads get time via the Thread Manager. Applications which are sleeping, and hence are not scheduled by the Process Manager, do not get their threads executed. Threads are per-application: when the application gets time, its threads get time.

Cooperative and preemptive threads are not given a priority and are scheduled in a round-robin fashion or as dictated by a "yield to" call or a custom scheduler. Both types of threads are guaranteed to begin execution in the normal operating mode of the application. Normal operating mode is defined as the addressing and CPU operation modes into which the application was launched. The operating mode will either be 24 or 32-bit MMU addressing mode, and user or supervisor CPU execution mode. At preemptive reschedule time, the addressing mode of the thread is restored to its preempted state. The CPU operating mode is not changed; rescheduling will only take place if the current thread is executing in the normal application CPU operating mode. If the normal operating mode of the CPU is user mode, and the current thread is executing in supervisor mode when preemption occurs, the Thread Manager does not reschedule and will return control back to the interrupted thread.

Cooperative threads get time when an explicit yield call is made to cause a context switch. All the rules that apply to WaitNextEvent or GetNextEvent hold true for yield calls across cooperative threads. For example, no assumptions can be made about the placement of unlocked handles.

Preemptive threads are not required to make yield calls to cause a context switch (although they certainly may) and share 50% of their CPU time with the currently executing cooperative thread. However, calling yield from a preemptive thread is desirable if that thread is not currently busy.

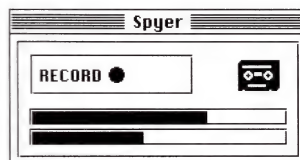
With the advent of multiply threaded applications comes the issue of data coherency. This is a problem where one thread of execution (either cooperative or preemptive) is looking at shared data while another thread is changing it. The Thread Manager provides a solution to this problem by providing the application with the ability to define a "critical" section of code which locks out preemption. With preemption disabled, a thread may look at or change shared or global data safely. The "critical" code mechanism is provided through the use of the ThreadBeginCritical and ThreadEndCritical calls. ThreadBeginCritical increments a counter semaphore and tells the Thread Manager to lock out the preemption mechanism. ThreadEndCritical does just the opposite - when the counter semaphore reaches zero, the preemption mechanism is re-enabled. ThreadBeginCritical/ThreadEndCritical pair provides developers with the building blocks needed for direct semaphore support.

Writing A Custom Thread Scheduler Routine: Preemption is disabled when the custom scheduler is called which prevents, among other things, reentrancy problems. There should be no yield or other scheduling calls made at this time. The custom

Spyer


What You Did Is What You See

If you are looking for a simple way to replay your actions - including mouse movement, for software testing or demonstration - Spyer is the answer.



Features:

- Records keyboard and mouse actions.
- Stores recorded data in files for future use.
- Controllable Replay speed.
- Works on any Macintosh application.
- A background process triggered by Hot Keys for easy operation.

	
Available:	19236
Recorded:	206
Replayed:	0

Price: \$ 39 + shipping



US: Tel: 718-991-1689 Fax: 718-328-8513 **ALink:** INCI.MISHA
International: Tel: 972-3-9613530 Fax: 972-3-9613827

POSITIONS WANTED

Available Immediately

1-800-736-3577

Expert 4D Programmers Less than 50¢ per hour!

More than 1,000 hours of development went into 4D Toolkit 2.0. With a one time price of \$395.00*, it's like hiring a crack team of 4D programmers at less than 50¢ per hour. Call 1-800-736-3577 to order your copy, or to receive a free demo.

4D TOOLKIT™

Options Computer Consulting
228 Bleecker Street #19
New York, NY 10014
TEL: 212-645-3577
FAX: 212-633-0336

* upgrade pricing available

scheduler is provided with a data record defining thread ID information which includes the size of the data record (for important future directions), the current thread ID, the suggested thread ID (which may be kNoThreadID), and the currently interrupted cooperative thread (or kNoThreadID). In addition to this information the custom scheduler must have knowledge about the threads it wishes to schedule. If the custom scheduler does not wish to select a thread it can pass back the suggested thread ID (or kNoThreadID) as the thread to schedule and let the Thread Manager's default scheduler decide. If the custom scheduler does not know about all of the threads belonging to the application (it may not if the system creates threads on behalf of the application), it should occasionally send back the suggested thread ID (or kNoThreadID) to give other threads a chance to be scheduled. Note that due to the round robin scheduling approach, the 'other' threads are not guaranteed to be next in line for scheduling. If the interrupted cooperative thread ID variable is not 'nil', the custom scheduler was called during the execution of a preemptive thread and must not schedule a cooperative thread other than the interrupted cooperative thread. Scheduling a cooperative thread at this time would effectively be causing cooperative thread preemption which could result in a system misunderstanding (crash).

Important: Scheduling with native threads is less complicated because there are no preemptive threads. Meaning the only way rescheduling happens is when a thread yields to any other thread.

Context: The default context of a thread consists of the CPU registers, the FPU registers if an FPU is present, and a few lowmem globals. Specifically, the saved data is as follows:

CPU Registers	FPU Registers
RD0 - RD7	FPCR, FPSR, FPIAR
RA0 - RA7	FP0 - FP7
SR (incl. CCR)	FPU frame

For power applications, the context looks something like this:

CPU Registers	FPU Registers	Machine Registers
R0-R31	FP0-FP31	CTR, LR, PC
	FPSCR	CR, XER

The thread context lives on a thread's A7 stack and the location of the thread context is saved at context switch time. The A5 register which contains a pointer to the application's "A5 world" and the initial thread MMU mode is initially set the same as the main application thread. This allows all threads to share in the use of the application's A5 world which gives threads access to open files and resource chains, for example. The MMU mode of a thread is saved away, and the mode of the interrupted thread is restored to allow preemption of threads which change the MMU operating mode. The FPU context is fully saved along with the current FPU frame.

Writing a Custom Thread Context Switching Routine: Preemption is disabled when the custom switching routine is called which prevents, among other things, reentrancy problems. There should be no yield or other scheduling calls made at this time. When a custom context switching routine is called, thread context is in transition, so calls to GetCurrentThread and uses of kCurrentThreadID will not be appropriate. Custom switching routines are defined on a per-thread in or out basis. Each thread is treated separately, which allows threads to mix and match custom switchers and parameters. A custom context switcher may be defined for entering a thread and another for exiting the same thread. Each context switching procedure is passed a parameter to be used at the application's discretion. For example, there could be one custom switching routine that is installed with a different parameter on each thread.

Note: If a custom thread switcher-inner is installed, it will be called before the thread begins execution at the thread entry point.

Important: The entire context is saved by ThreadsLib for any native application. This is due to the fact that compilers can use all the registers during optimization, even the floating point ones.

Programmatic Interface

Data Types

The Thread Gestalt selector and bit field definition are used to determine if the threads package is installed. The gestaltThreadsPresent bit in the result will be true if the Thread Manager is installed. Other bits in the result field are reserved for future definition.

```
CONST
gestaltThreadMgrAttr= 'thds';  {Thread Manager attributes}
gestaltThreadMgrPresent= 0;  {bit true if Threads present}
gestaltSpecificMatchSupport= 1; {bit true if 'exact match' API supported}
gestaltThreadsLibraryPresent= 2; {bit true if ThreadsLib is present}
```

The ThreadState data type indicates the general operational status of a thread. A thread may be waiting to execute, suspended from execution, or executing.

```
TYPE
ThreadState= INTEGER;
```

```
CONST
kReadyThreadState = 0;  {thread is eligible to run}
kStoppedThreadState = 1; {thread is not eligible to run}
kRunningThreadState = 2; {thread is running}
```

The ThreadTaskRef is used to allow calls to the Thread Manager at a time when the application context is not necessarily the current context.

```
TYPE
ThreadTaskRef= Ptr;
```

The ThreadStyle data type indicates the broad characteristics of a thread. A cooperative thread is one whose execution environment is sufficient for calling Toolbox routines (this requires a larger stack, for example). A preemptive thread is one that does not need to explicitly yield control, and executes

preemptively with all other threads.

```
TYPE
  ThreadStyle= LONGINT;

CONST
  kCooperativeThread = 1; {thread can use Macintosh Toolbox}
  kPreemptiveThread = 2;  {thread doesn't necessarily yield}
```

Note: kPreemptiveThread is not defined for use with power Thread Manager.

The ThreadID data type identifies individual threads. ThreadIDs are unique within the scope of the application process. There are a few pre-defined symbolic thread IDs to make the interface easier.

```
TYPE
  ThreadID = LONGINT;
```

```
CONST
  kNoThreadID= 0; {no thread at all}
  kCurrentThreadID = 1; {thread whose context is current}
  kApplicationThreadID= 2; {thread created for app at launch}
```

The ThreadOptions data type specifies options to the NewThread routine.

```
TYPE
  ThreadOptions= LONGINT;
```

```
CONST
  kNewSuspend = 1; {begin new thread in stopped state}
  kUsePremadeThread = 2; {use thread from supply}
  kCreateIfNeeded = 4; {allocate if no premade exists}
  kFPUNotNeeded = 8; {don't save FPU context}
  kExactMatchThread = 16; {force exact match over best fit}
```

The Following information is supplied to a custom scheduler.

Note: kFPUNotNeeded is ignored by the power Thread Manager because floating point registers are always saved.

```
TYPE
  SchedulerInfoRecPtr = ^SchedulerInfoRec;
  SchedulerInfoRec = RECORD
    InfoRecSize:      LONGINT;
    CurrentThreadID:  ThreadID;
    SuggestedThreadID: ThreadID;
    InterruptedCoopThreadID: ThreadID;
  END;
```

The following are the type definitions for a thread's entry routine, a custom scheduling routine, custom context switching routine, and a thread termination routine.

```
TYPE
  ThreadEntryProcPtr = ProcPtr; {entry routine}
  { FUNCTION ThreadMain (threadParam: LONGINT): LONGINT; }

  ThreadSchedulerProcPtr= ProcPtr; {custom scheduler}
  { FUNCTION ThreadScheduler (schedulerInfo: SchedulerInfoRec): ThreadID; }

  ThreadSwitchProcPtr = ProcPtr; {custom switcher}
  { PROCEDURE ThreadSwitcher (threadBeingSwitched: ThreadID;
    switchProcParam: LONGINT); }

  ThreadTerminationProcPtr = ProcPtr; {custom switcher}
  { PROCEDURE ThreadTerminator (threadTerminated: ThreadID;
    terminationProcParam: LONGINT); }
```

The following are the type definitions to allow a debugger to

watch the creation, deletion and scheduling of threads on a per-application basis.

```
TYPE
  DebuggerNewThreadProcPtr = ProcPtr;
  { PROCEDURE DebuggerNewThread (threadCreated: ThreadID); }

  DebuggerDisposeThreadProcPtr = ProcPtr;
  { PROCEDURE DebuggerDisposeThread (threadCreated: ThreadID); }

  DebuggerThreadSchedulerProcPtr = ProcPtr;
  { FUNCTION DebuggerThreadScheduler(schedulerInfo: SchedulerInfoRec):
    ThreadID; }
```

The following are Thread Manager specific errors.

```
CONST
  threadTooManyReqsErr= -617;
  threadNotFoundErr   = -618;
  threadProtocolErr   = -619;
```

General Purpose Routines

These routines allow the application to create, initiate, and delete threads.

```
FUNCTION CreateThreadPool (threadStyle: ThreadStyle;
  numToCreate: INTEGER; stackSize: Size):OSErr;
```

CreateThreadPool creates a specified number of threads having the given style and stack requirements. The thread structures are put into a supply for later allocation by the NewThread routine. This function may be called repeatedly, which will add threads to the one application thread pool. A pool of threads may be needed if, for example, preemptive threads need to spawn threads. Preemptive threads may only create new threads from an existing thread pool; this is to prevent Toolbox reentrancy if memory allocation must be made to satisfy the request.

If not all of the threads could be created, none are allocated (it's all or nothing!).

Note: Threads in the allocation pool can not be individually targeted by any of the Thread Manager routines (i.e. they are not associated with ThreadIDs). The only routines that refer to threads in the allocation pool are NewThread and GetFreeThreadCount, but they address the application pool as a whole.

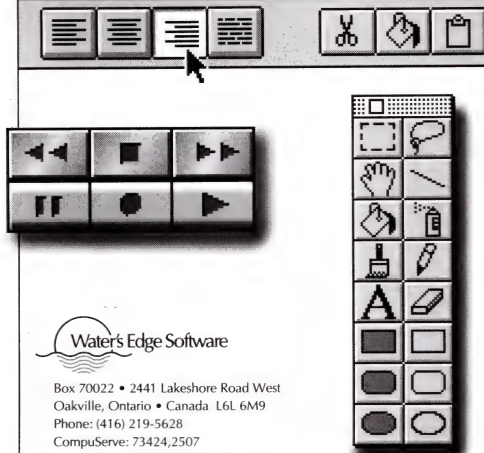
Note: The stackSize parameter is the requested stack size for this set of pooled threads. This stack must be large enough to handle saved thread context, normal application stack usage, interrupt handling routines and CPU exceptions. By passing in a stackSize of zero <0>, the Thread Manager will use its default stack size for the type of threads being created. To determine the default stack size for a particular thread type, see the ThreadDefaultStackSize routine for more information.

Result codes:	noErr	Specified threads were created and are available
	memFullErr	Insufficient memory to create the thread structures
	paramErr	Unknown threadStyle, or using kPreemptiveThread with the power Thread Manager

```
FUNCTION GetFreeThreadCount (threadStyle: ThreadStyle;
  VAR freeCount: INTEGER):OSErr;
```

GetFreeThreadCount finds the number of threads of the given thread style that are available to be allocated. The number of available threads is raised by a successful call to

Simply the best GUI Building/Event Managing libraries



Tools Plus™ 2.5

Tools Plus gives you the routines you need to create a professional looking user interface. Then we make it work. It's that simple.

- For THINK C and Symantec C/C++ (5.0.4 and later) or THINK Pascal
- Over 170 high-powered "set and forget" routines that automate and enhance: event handling, windows, the tool bar, floating palettes, cursors, buttons, picture buttons, scroll bars, menus (pull-down, hierarchical and pop-up), list boxes, fields, Edit menu, clipboard, Dynamic Alerts, and more...
- Easy to learn and easy to use
- Substantial code reduction
- Dramatic code simplification
- Significantly less debugging
- System 6 and 7 compatible
- For novice, intermediate and advanced programmers
- Runs fast; needs little memory or disk space
- Safer than toolbox routines
- No royalties

Language

- ☐ C/C++
☐ Pascal
☒ Both

- ☒ Saves Time
☒ Saves Money
☒ Easy to Use

Free Evaluation Kit:

CompuServe GO MACDEV,
 C & Pascal library, file name: TP252.SEA
 AppleLink Software Sampler/Software
 Collection/Programmer Tools/Tools Plus
 Disk also available by mail.

OK

Water's Edge Software

Box 70022 • 2441 Lakeshore Road West
 Oakville, Ontario • Canada L6L 6M9
 Phone: (416) 219-5628
 CompuServe: 73424.2507
 Internet: 73424.2507@CompuServe.com

Tools Plus for C/C++ or Pascal only \$149 US or \$199 US for both.
 (We accept VISA and American Express. Add \$10 for shipping.)

CreateThreadPool or DisposeThread (with the recycleThread parameter set to "true"). The number is lowered by calls to NewThread when a pre-made thread is allocated.

Result codes: noErr freeCount has the count of available threadStyle threads
 paramErr Unknown threadStyle, or using kPreemptiveThread with the power Thread Manager

```
FUNCTION GetSpecificFreeThreadCount (threadStyle: ThreadStyle;
    stackSize: Size; VAR freeCount: INTEGER):OSErr;
```

GetSpecificFreeThreadCount finds the number of threads of the given thread style and stack size that are available to be allocated. The number of available threads is raised by a successful call to CreateThreadPool or DisposeThread (with the recycleThread parameter set to "true"). The number is lowered by calls to NewThread when a pre-made thread is allocated.

Result codes: noErr freeCount has the count of available threadStyle threads
 paramErr Unknown threadStyle, or using kPreemptiveThread with the power Thread Manager

```
FUNCTION GetDefaultThreadStackSize (threadStyle: ThreadStyle;
    VAR stackSize: Size):OSErr;
```

GetDefaultThreadStackSize returns the default stack needed for the type of thread requested. The value returned is the stack size used if zero <0> is passed into the CreateThreadPool & NewThread stackSize parameter. This value is by no means absolute, and most threads do not need as much stack space as the default value. This and the ThreadCurrentStackSize routines are provided to help tune your threads for optimal memory usage.

Result codes: noErr stackSize has the default stack needed for threadStyle threads
 paramErr Unknown threadStyle, or using kPreemptiveThread with the power Thread Manager

```
FUNCTION ThreadCurrentStackSize (thread: ThreadID;
    VAR freeStack: LONGINT):OSErr;
```

ThreadCurrentStackSize returns the current stack space

available for the desired thread. Be aware that various system services will run on your thread stack (interrupt routines, exception handlers, etc.) so be sure to account for those in your stack usage calculations. See GetDefaultThreadStackSize for more information.

Result codes: noErr freeCount has the amount of stack space available in thread
 threadNotFoundErr There is no existing thread with the specified ThreadID

Note: When using this routine from a preemptive thread, care must be taken when obtaining information about another thread. It is not always possible to know the stack environment of a preempted thread – the Toolbox may have temporarily changed stacks to perform its functions.

```
FUNCTION NewThread (threadStyle: ThreadStyle;
    threadEntry: ThreadEntryProcPtr;
    threadParam: LONGINT;
    stackSize: Size;
    options: ThreadOptions;
    threadResult: LongIntPtr;
    VAR threadMade: ThreadID):OSErr;
```

NewThread creates or allocates a thread structure with the specified characteristics, and puts the thread's identifier in the threadMade parameter. The threadEntry parameter is the entry address of the thread, and is best represented as a Pascal-style function. The threadParam parameter is passed as a parameter to that function for application defined uses. When the thread terminates, the function result is put into threadResult (pass nil for threadResult if you are not interested in the thread's result). In the case of an error returned, the threadMade parameter is set to kNoThreadID.

The ThreadOptions parameter specifies optional behavior of NewThread. Thread options are summed together to create the desired combination of options. The kNewSuspend option indicates that the new thread should begin in the

kStoppedThreadState, ineligible for execution. The kUsePremadeThread option requests that the new thread be allocated from an existing pool of premade threads. By default, threads allocated from the thread pool are done so on a stack size best fit basis. The kExactMatchThread option requires threads allocated from the pool to have a stack size which exactly matches the stack size requested by NewThread. The kCreateIfNeeded option gives NewThread permission to allocate an entirely new thread if the supply allocation request can not be honored. The kFPUNotNeeded option will prevent FPU context from being saved for the thread. This option will speed the context switch time for a thread that does not require FPU context.

Important: The storage for threadResult needs to be available when the thread terminates. Therefore, an appropriate storage place would be in the application globals or as local variable to the application's main routine. An inappropriate place would be as a local variable to a subroutine that completes before the thread terminates.

Important: Preemptive threads may only call this routine if the kUsePremadeThread option is set.

Note: The stackSize parameter is the requested stack size of the new thread. This stack must be large enough to handle saved thread context, normal application stack usage, interrupt handling routines and CPU exceptions. By passing in a stackSize of zero <0>, the Thread Manager will use its default stack size for the type of threads being created. To determine the default stack size for a particular thread type, see the ThreadDefaultStackSize routine for more information.

Note: ThreadsLib will not allow you to create preemptive threads, as well as it ignores the kFPUNotNeeded option, since all of the native context has to be saved.

Result codes:	noErr	Specified thread was made or allocated
	memFullErr	Insufficient memory to create the thread structure
	threadTooManyRegsErr	There are no matching thread structures available
	paramErr	Unknown threadStyle, or using kPreemptiveThread with the power Thread Manager

```
FUNCTION DisposeThread (threadToDump: ThreadID; threadResult:
LONGINT; recycleThread: BOOLEAN):OSErr;
```

DisposeThread gets rid of the specified thread. The threadResult parameter is passed on to the thread's creator (see NewThread). The recycleThread parameter specifies whether to return the thread structure to the allocation pool supply, or to free it entirely.

Result codes:	noErr	Specified thread was disposed
	threadNotFoundErr	There is no existing thread with the specified ThreadID
	threadProtocolErr	ThreadID specified the application thread

Note: Disposing a thread from a preemptive thread will force the disposed thread to be recycled regardless of the recycleThread setting. Returning from a thread causes itself to be disposed.

Basic Scheduling Routines

These routines allow the application to get information

about and have basic scheduling control of the current thread, without specific attention to the other threads in the application.

```
FUNCTION GetCurrentThread (VAR currentThreadID: ThreadID):OSErr;
```

GetCurrentThread finds the ThreadID of the current thread, and stores it in the currentThreadID parameter.

Result codes:	noErr	current ThreadID returned
	threadNotFoundErr	There is no current thread

```
FUNCTION YieldToAnyThread : OSErr;
```

YieldToAnyThread relinquishes the current thread's control, causing generalized rescheduling. The current thread suspends in the kReadyThreadState, awaiting availability of the CPU. When the thread is again scheduled, this routine regains control and returns to the caller.

Important: Threads must yield in the CPU addressing mode (24 or 32-bit) in which the computer normally operates. However, threads may be preempted in any CPU addressing mode.

Result codes:	noErr	Current thread has yielded and is now running again.
	threadProtocolErr	Current thread is in a critical section (see ThreadBeginCritical)

Preemptive Thread Scheduling Routines

These routines are useful when the application includes preemptive threads.

```
FUNCTION ThreadBeginCritical : OSErr;
```

ThreadBeginCritical indicates to the Thread Manager that the current thread is entering a critical section with respect to all other threads in the current application. This prevents preemptive scheduling to prevent interference from the other threads. Note that this routine is not needed if there are no active preemptive threads in the application.

Note: Critical sections may be nested.

Important: Preemptive threads may be interrupted to execute a cooperative thread, so critical sections can exist in them, as well.

Result codes:	noErr	Current thread can now execute critical section
---------------	-------	---

```
FUNCTION ThreadEndCritical : OSErr;
```

ThreadEndCritical indicates to the Thread Manager that the current thread is exiting a critical section.

Result codes:	noErr	Current thread is now out of most nested critical section
	threadProtocolErr	Current thread is not in a critical section (see ThreadBeginCritical)

Advanced Scheduling Routines

These routines allow the application to schedule threads with a greater control and responsibility. Typically, an application-wide view of threads is needed when applying these routines.

Pascal → C++

Stuck with Pascal source code? Want to move to C++?

OP2CPlus is a Macintosh tool for converting Object Pascal source code to C++ source code. It has already been used to translate hundreds of thousands of lines of Object Pascal to C++.

Ease your transition to OpenDoc or PowerPC

- ☐ Convert in days instead of months
- ☐ Generates C++ classes
- ☐ Works with MacApp 2 or 3
- ☐ Translates Think or MPW Pascal
- ☐ Full ANSI C source code included
- ☐ Just \$895

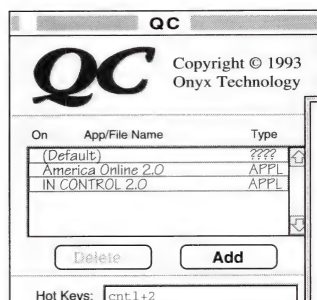
Graphic Magic Inc, 180 Seventh Ave, Suite 201
Santa Cruz CA 95062 Tel (408) 464 1949
Fax (408) 464 0731 AppleLink GRAPHICMAGIC

QC

NOW SHIPPING

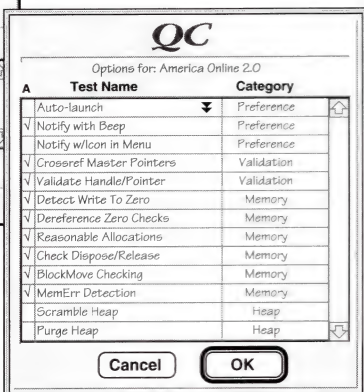
QC: the Macintosh Testing solution.
Subject your code to **brutal** stress conditions to **make it break** consistently.

Or use QC during the development cycle to casually detect block boundary overwrites, invalid BlockMoves, writes to location zero, and more...**saving countless hours of needless debugging.**



Features:

- Works with any program without modification - source not required
- Easy to use: just hit the hotkey.
- Fast heap scramble and purge
- Invalidates all free memory
- Detects runtime block overwrites
- Warns of DisposHandle on resource and ReleaseResource on handles.
- Validates BlockMove destinations
- Sophisticated heap verification
- Powerful API for precision control



30 DAY
NO QUESTIONS
ASKED
MONEY BACK
GUARANTEE



ONYX TECHNOLOGY
7811 27th Ave W.
Bradenton, FL 34209

\$99

SPECIAL
INTRODUCTORY
PRICE

Tel: 813.795.7501 Fax: 813.792.5152 AOL: OnyxTech ALink: D2236 CIS: 70550,1377

```
FUNCTION YieldToThread (suggestedThread: ThreadID):OSErr;
```

YieldToThread relinquishes the current thread's control, causing generalized rescheduling, but passes the suggestedThread to the scheduler. The current thread suspends in the kReadyThreadState, awaiting availability of the CPU. When the thread is again scheduled, this routine regains control and returns to the caller.

Important: Threads must yield in the CPU addressing mode (24 or 32-bit) in which the computer normally operates. Preemptive threads should never explicitly yield to cooperative threads. Doing so would in effect be causing preemption between cooperative threads.

Result codes: noErr Current thread has yielded and is now running again.
threadNotFoundErr There is no existing thread with the specified id,
or the suggested thread is not in the ready state.
threadProtocolErr Current thread is in a critical section (see
ThreadBeginCritical)

```
FUNCTION GetThreadState (threadToGet: ThreadID;  
VAR threadState:ThreadState):OSErr;
```

In the presence of preemptive threads, the state of any thread can change asynchronously (at any time). This implies that the value returned from GetThreadState might be inaccurate by the time the caller checks it. If absolute correctness is required, this call should be made while preemptive scheduling is disabled, such as in a critical section (delimited by ThreadBeginCritical & ThreadEndCritical) or during the custom scheduling routine (see SetThreadScheduler).

Result codes: noErr threadState contains the specified thread's state
threadNotFoundErr There is no existing thread with the specified ThreadID

```
FUNCTION SetThreadState (threadToSet: ThreadID;  
newState:ThreadState; suggestedThread: ThreadID):OSErr;
```

SetThreadState puts the specified thread into the specified state. If the current thread is specified, and newState is either kReadyThreadState or kStoppedThreadState, rescheduling occurs and suggestedThreadID is passed on to the scheduler.

Important: Threads must yield in the CPU addressing mode (24 or 32-bit) in which the computer normally operates.

Result codes: noErr Thread was put in the specified state. If this
current thread, it is now running again
threadNotFoundErr There is no existing thread with the specified
ThreadID, or the suggested thread is not in the
ready state.
threadProtocolErr Caller attempted to suspend/stop the desired thread,
but the desired thread is in a critical section (see
ThreadBeginCritical), or newState is an invalid state.

```
FUNCTION SetThreadStateEndCritical (threadToSet: ThreadID;  
newState:ThreadState; suggestedThread: ThreadID):OSErr;
```

SetThreadStateEndCritical atomically puts the specified thread into the specified state and exits the current thread's critical section. If the current thread is specified, and newState is either kReadyThreadState or kStoppedThreadState, rescheduling occurs and suggestedThreadID is passed on to the scheduler. This call is useful in cases where the current thread needs to put itself in a stopped state at the end of a critical section,

thereby closing the scheduling window between a call to ThreadEndCritical and SetThreadState.

Important: Threads must yield in the CPU addressing mode (24 or 32-bit) in which the computer normally operates.

Result codes:	noErr	Thread was put in the specified state. If this was the current thread, it is now running again
	threadNotFoundErr	There is no existing thread with the specified ThreadID, or the suggested thread is not in the ready state.
	threadProtocolErr	Current thread is not in a critical section (see ThreadBeginCritical), or newState is an invalid state.

```
FUNCTION GetThreadCurrentTaskRef ( VAR threadTRef:
                                ThreadTaskRef ):OSErr;
```

GetThreadCurrentTaskRef returns an application process reference for later use, potentially at interrupt time. This task reference will allow the Thread Manager to get & set information for a particular thread during any application context.

Result codes:	noErr	Thread task reference was returned
---------------	-------	------------------------------------

```
FUNCTION GetThreadStateGivenTaskRef ( threadTRef: ThreadTaskRef;
threadToGet: ThreadID; VAR threadState: ThreadState ):OSErr;
```

GetThreadStateGivenTaskRef returns the state of the given thread in a particular application. The primary use of this call is for completion routines or interrupt level code which must acquire the state of a given thread at times when the application context is unknown.

Result codes:	noErr	threadState contains the specified thread's state
	threadNotFoundErr	There is no existing thread with the specified ThreadID & TaskRef
	threadProtocolErr	Caller passed in an invalid TaskRef.

```
FUNCTION SetThreadReadyGivenTaskRef( threadTRef: ThreadTaskRef;
threadToSet: ThreadID ):OSErr;
```

SetThreadReadyGivenTaskRef will mark a stopped thread as ready and eligible to run, but will not be put in the ready queue until the next time rescheduling occurs. Threads marked as stopped are the only types eligible to be marked as ready by this routine. An example use of this routine is to allow a completion routine to unblock the thread which stopped itself after making an asynchronous I/O call.

Result codes:	noErr	The specified thread is marked as ready.
	threadNotFoundErr	There is no existing thread with the specified ThreadID & TaskRef
	threadProtocolErr	Caller attempted to mark a thread ready that is not in the stopped state, or caller passed in an invalid TaskRef.

```
FUNCTION SetThreadScheduler (threadScheduler:
                             ThreadSchedulerProcPtr):OSErr;
```

SetThreadScheduler installs a custom thread scheduler, replacing any current custom scheduler. A threadScheduler of nil specifies "none".

Important: The application A5 global pointer is not guaranteed to be the value in the CPU's A5 register when the Thread Manager calls back to your custom scheduler. Be sure to set up register A5 before accessing global data.

Result codes:	noErr	Specified scheduler was installed
---------------	-------	-----------------------------------

Programmer Training

MACINTOSH SEMINARS & CONSULTING

Richey Software Training provides professional, *customized* programming seminars and industry consulting.

Rich content & hands-on lab exercises shorten your learning curve. On-site training is convenient — your team eliminates expensive travel costs and consuming down-time.

“ Professional training & consultancy that really hits the mark. ”

Call today to find out how Richey Software Training delivers professional seminars and consulting nationwide.



Training Mac Programmers Since 1986

707-869-2836

AppleLink: RICHEY.SOFT

INTERNET: 70413.2710@compuserve.com

P.O. BOX 1809, GUERNEVILLE, CA 95446-1809

© 1994 Richey Software Training. All trademarks or registered trademarks are the property of their respective owners. Specifications subject to change.

MAC SEMINARS

- C & C++
- OOP
- MacApp
- PowerPC
- AppleScript
- MPW
- System 7
- Debugging
- SourceBug

Put A Spelling Checker In Your Application

Working Software offers several options for adding a spelling checker to your applications. Call us.

THE FREE WAY – Add **free** Apple Events Word Services and our award-winning Spellswell 7 will work with your app as if built-in. (WordPerfect, Omnis 7, Eudora & InfoDepot use this.) Our **FREE** Word Services Developer Kit includes source code for a sample word processor to guide you.

THE CUSTOM WAY – Compile our OEM speller (MPW, Think C or Pascal) into your **Mac and/or Windows** application (takes 1–2 days). Legal, Medical & other dictionaries available. Fees depend on your circumstances - call for details.

Contact us for **FREE** Object-Only Demo and/or **FREE** Word Services Software Development Kit.

Working Software, Inc.

P.O. Box 1844 / Santa Cruz, CA 95061-1844
(800) 229-9675 / (408) 423-4596 / FAX (408) 423-5699
AppleLink D0140 / CompuServe 76004,2072

4 out of 5 programmers who chew gum recommend new QUED/M 2.7

The American Dental Association can't explain it. But gum chomping Mac programmers everywhere, and even those that don't chew, are raving about the sheer power and convenience QUED/M™ brings to their text editing. And version 2.7 makes it even easier to chew gum and program at the same time, with these powerful new features:

- ☛ Integrated support for THINK Project Manager™ 6.0 & 7.0
- ☛ THINK™ debugger support
- ☛ CodeWarrior™ support (now it's easier than ever to program for the Power Macintosh®!)
- ☛ MPW ToolServer™ support
- ☛ PopUpFuncs™ support
- ☛ Frontier™ Do Script support

Of course, QUED/M 2.7 still has all the features that make it easier to use than any other text editor:

- ☛ Macro Language lets you automate tedious tasks
- ☛ Search and Replace through multiple unopened files and using GREP metacharacters
- ☛ File comparisons using GNU Diff
- ☛ Unlimited undos and redos
- ☛ 10 Clipboards that can be edited, saved, and printed
- ☛ Customizable menu keys
- ☛ Text folding
- ☛ Display text as ASCII codes
- ☛ Plus many more features!

Find out more. Call 800-309-0355 today!

Here's something to chew on: Try QUED/M 2.7 now for just \$69! You'll get a 30-day money back guarantee too! Call now to order. 800-309-0355.

QUED/M is a trademark of Nisus Software Inc. All other products are trademarks or registered trademarks of their respective holders.

107 S. Cedros Ave. • Solana Beach, CA 92075 • Tel (619) 481-1477 • Fax (619) 481-6154

NISUS
Software Inc.

```
FUNCTION SetThreadSwitcher (thread: ThreadID;
  threadSwitcher: ThreadSwitchProcPtr;
  switchProcParam: LONGINT; inOrOut: BOOLEAN):OSErr;
```

SetThreadSwitcher installs a custom thread context switching routine for the specified thread in addition to the standard processor context which is always saved. A threadSwitcher of nil specifies "none". The inOrOut parameter indicates whether the routine is to be called when the thread is switched in (inOrOut is "true"), or when the thread is switched out (inOrOut is "false"). The switchProcParam specifies a parameter to be passed to the thread switcher.

Each thread is treated separately, so threads are free to mix and match custom switchers and parameters. For example, there could be one custom switching routine that is installed with a different parameter on each thread.

Important: The application A5 global pointer is not guaranteed to be the value in the CPU's A5 register when the Thread Manager calls back to your custom switcher. Be sure to set up register A5 before accessing global data.

Result codes: noErr Specified thread switcher was installed
 threadNotFoundErr There is no existing thread with the specified ThreadID

```
FUNCTION SetThreadTerminator (thread: ThreadID;
  threadTerminator: ThreadTerminationProcPtr;
  terminationProcParam: LONGINT):OSErr;
```

SetThreadTerminator installs a custom thread termination

routine for the specified thread. The custom thread termination routine will be called at the time a thread is exited or is manually disposed of. The terminationProcParam specifies a parameter to be passed to the thread terminator.

Each thread is treated separately, so threads are free to mix and match custom terminators and parameters. For example, there could be one custom termination routine that is installed with a different parameter on each thread.

Result codes: noErr Specified thread terminator was installed
 threadNotFoundErr There is no existing thread with the specified ThreadID

Thread Debugging Support

The following routine is set aside for debuggers to install watchdog procedures when the major state of a thread changes. These routines are reserved for use by debuggers to help in the development of multithreaded applications.

```
FUNCTION SetDebuggerNotificationProcs (
  notifyNewThread: DebuggerNewThreadProcPtr;
  notifyDisposeThread: DebuggerDisposeThreadProcPtr;
  notifyThreadScheduler: DebuggerThreadSchedulerProcPtr
):OSErr;
```

SetDebuggerNotificationProcs sets the per-application support for debugger notification of thread birth, death and scheduling. The debugger will be notified with the threadID of the newly created or disposed of thread. The debugger is also notified if the thread

simply returns from its highest level of code and thus automatically disposes itself. The DebuggerThreadSchedulerProcPtr will be called after the custom scheduler and the Thread Manager's generic scheduler have decided on a thread to schedule. In this way, the debugger gets the last shot at a scheduling decision.

Important: All three debugger callbacks are installed when this call is made. It is not possible to set one or two of the callbacks at a time with this routine, and it is not possible to chain these routines. This restriction ensures that the last caller of this routine owns all three of the callbacks. Setting a procedure to NIL will effectively disable it from being called. Also note that the application A5 global pointer is not guaranteed to be the value in the CPU's A5 register when the Thread Manager calls back to your debugger procedures.

Result codes: noErr Debugger procs have been installed

Routines That Move Or Purge Memory:

CreateThreadPool
NewThread
DisposeThread - When 'recycleThread' is false

Routines You Can Call During Preemptive Thread Execution:

NewThread - When 'kUsePremadeThread' is used
DisposeThread - When 'recycleThread' is true
GetCurrentThread
GetFreeThreadCount
GetDefaultThreadStackSize
ThreadCurrentStackSize
GetThreadState
SetThreadState - See note below
SetThreadStateEndCritical - See note below
ThreadBeginCritical
ThreadEndCritical
YieldToAnyThread
YieldToThread - See note below
SetThreadScheduler
SetThreadSwitcher
SetDebuggerNotificationProcs
GetThreadCurrentTaskRef

Note: The SetThreadState, SetThreadStateEndCritical, and YieldToThread routines are usable during preemptive execution only when suggestedThread parameter is either kNoThreadID or is another preemptive thread. Explicitly requesting a cooperative thread to run from a preemptive thread is dangerous and should be avoided.

Routines You Can Call At Interrupt Time:

GetThreadStateGivenTaskRef
SetThreadReadyGivenTaskRef

Toolbox & OS Routines You Can Call From a Cooperative Thread:

- All routines are available from a cooperative thread after MaxApplZone has been called.
- On a Mac Plus, only the main application thread may make resource manager calls (explicitly UpdateResFile & CloseResFile).

Thread Manager... continued on page 68

Dave Wilson's

QuickApp™ **Learn Framework** **Programming!**



Everyone is moving into object oriented programming. You've heard all about these great ideas of code reuse and rapid applications development. You looked at MacApp, TCL, and PowerPlant, but they're just too complex. Wasn't the idea of frameworks to make your life easier?

Now there is QuickApp. A C++ app framework designed for **small jobs**. QuickApp provides you with the basics you need from your app framework, but little else. This means that your compile and link steps happen **fast**. QuickApp was built to get you up to speed quickly. It even comes with a code generator to write your boiler plate for you.

QuickApp has been used to teach hundreds of people about OOP. QuickApp is used as the basis for Apple Computer's Object Oriented Fundamentals 5-day seminar. QuickApp is a great way to get your feet wet without drowning. All the ideas you'll learn while using QuickApp can be applied to using other frameworks such as MacApp.

Requires Symantec C++ or Code Warrior C++. When used with Code Warrior QuickApp can build native Power Macintosh applications. Includes full Source Code.

Call Today!
Emergent Behavior™

635 Wellsbury Way
Palo Alto, CA 94306
(415) 494-6763
AppleLink: Wilson6

Price: \$149
+shipping

Visa / Mastercard Accepted



By Scott T Boyd, Editor

Many of you have asked for pointers to interesting places. We've whipped up a quick hodge-podge of points of interest. It's certainly incomplete. If there's something you'd like to see here, please drop us a note at editorial@xplain.com.

In case you're not familiar with Universal Resource Locator (URL) format, it's essentially `<servicekind>://<servername>/<pathname>`. That's a bit of an oversimplification (e.g. the path can be far more interesting), but it should be enough to get you started.

Products mentioned in this issue

MacTech Magazine	ftp://ftp.netcom.com/pub/xplain
SmalltalkAgents	see inside back cover
Object Master Universal	see page 8
Resorcerer	see page 1
Prograph CPX	(800) 927-4847 or (902) 455-4446 voice (902) 455 2246 fax

Interesting Places

Lisp	http://www.cs.rochester.edu/u/miller/alu.html
Hawaiian fonts	http://www.mauui.com/~mauilink
Smalltalk	http://www.qks.com/
OpenDoc	ftp://cil.org
alt.sources.mac archive	ftp://ftp.bio.bgsu.edu/
Best of the Net www sites	http://nearnet.gnn.com/gnn/gnn.html
MacGL	ftp://ftp.netcom.com/pub/loceff

the OpenGL implementation

People/Places

Apple	ftp://ftp.apple.com/
Apple developer www	http://www.support.apple.com
Bill Modesitt	ftp://ftp.mauui.com/pub/mauisw
Maui High Performance Computer Center	http://www.mhpcc.edu/mhpcc.html
<i>There are also some great pictures of Maui!</i>	
Consensus	http://www.consensus.com:8300

Peter Lewis products	nic.switch.ch/mac/software/peterlewis/ amug.org/pub/peterlewis/ redback.cs.uwa.edu.au/others/peterlewis/ ftp.sri.ucl.ac.be/pub/
----------------------	--

French versions

Essential tools for getting on and cruising the net. *Anarchie* (pronounced *anarchy*) is a great ftp browser and downloader, as well as a great tool for finding software by name. *Anarchie* exemplifies much of what more Mac software should include: asynchronous i/o, threaded, and great drag and drop support. For example, you can drag a file from an ftp server to your desktop while you're waiting for a search to complete and while you're getting a directory listing from another ftp server. Peter has written other interesting pieces of net software, as well, including MacTCP Watcher, FTPd, Mungelimage, Daemon, TCP2Serial, and others. *ObiWan* is a help system he's written. Most of these are free or are inexpensive shareware.

Getting on the net

For those of you with only unix shell account network access, you may want to check out TIA, The Internet Adapter. The Internet Adaptor is a program for unix systems that lets you emulate a SLIP line in software, and that lets you run software like Mosaic and NewsWatcher. Telnet to tia.marketplace.com or point your www browser at marketplace.com for more info.



Thread Manager... continued from page 67

Toolbox & OS routines You Can Call From a Preemptive Thread:

Preemptive threads must follow the same rules as interrupt service routines as to which Toolbox and OS calls they may make.

PART 3 – Gotchas & Bugs

Gotcha: The Memory Manager routine `MaxApplZone` must be called before any thread other than the main application thread allocates memory, or causes memory to be allocated. See *Inside Macintosh Memory* for information on using memory and expanding the application heap.

Gotcha: Making certain calls to the Toolbox & OS during preemptive thread execution is a programming error; calls which may not be made at interrupt time may not be made by a preemptive thread. This includes calls to `LoadSeg` which get made on behalf of the application when accessing code segments which have not yet been loaded into memory. Applications must be sure that all code segments used by preemptive threads are preloaded and do not get unloaded.

One method of insuring certain traps do not get called at the wrong time is to define custom context switchers for preemptive threads. A custom context switcher-inner could be written to save and change the trap address of the trap in question (say `LoadSeg`) to a routine which drops into the debugger if that trap gets called while the thread is switched in. A custom context switcher-outer would then restore the original trap address for the rest of the application.

Gotcha: On a Mac Plus only, the main application thread should be the only thread which makes use of the Resource Manager. Specifically, calls to `UpdateResFile` or `CloseResFile` should only be made by the main application thread on a Mac Plus. All other Macintoshes support Resource Manager calls from any cooperative thread.

Gotcha: The application A5 global pointer is not guaranteed to be the value in the CPU's A5 register when the Thread Manager calls back to your custom call back routines. Be sure to set up register A5 before accessing global data from a custom scheduler, custom switchers, termination procedures, and debugger call back routines.





By Kurt Schmucker, Apple Computer, Inc.

Prograph CPX - A Tutorial

OOPS! Where'd they put all the semicolons?

This article is an abridged version of a chapter in the forthcoming book "Application Frameworks," edited by Ted Lewis and to be published later this year by Manning Publications/Prentice-Hall. Reprinted with permission.

Prograph CPX is a commercially-supported, self-contained, object-oriented programming language and development environment running on the Macintosh. It can produce small footprint, stand-alone applications, and is packaged with an application framework – called the Prograph ABCs (Application Builder Classes). The ABCs are roughly comparable in the breadth and depth of their functionality to other Macintosh frameworks like MacApp [Schmucker, 1988] and TCL [Parker, 1993]. While we'll touch on all three aspects of Prograph – the language, the environment, and the application framework – the major emphasis will be on the framework, its structure, and its use.

"Prograph integrates four key trends emerging in computer science:

- Prograph is a visual programming language
- Prograph is object-oriented
- Prograph supports dataflow specification of program execution
- Prograph provides an object-oriented application building toolkit."

– from the Prograph 2.5 manual, page 1

Prograph offers these unique features:

- its language: an object-oriented, visual, dataflow language,
- tight integration of the language with the editor, interpreter, and debugger into a single unified development tool
- several aspects of the ABCs, including the ability to package classes with custom graphical editors for their instances.

Prograph CPX is the principal product of Prograph International, and has been shipping on the Macintosh since 1988. It is positioned as a development tool that has the interactive 'feel' of the Macintosh Finder, yet with all the power to access any Macintosh internal that can be accessed from any C, C++, or Pascal programming tool. In fact, it would be reasonable to state that Prograph is the first industrial-strength visual programming system.

Prograph users run the full gamut of Macintosh application developers: from former HyperCard scripters or 4D developers, to those who are quite comfortable using MacApp or TCL in C++, and from developers who are primarily concerned with the functionality of their product (application developers) to those who are more concerned with the data content of their work (title developers). They have used Prograph to implement shrink-wrapped products, in-house applications, prototypes, and database front-ends, among other uses.

As of this writing (September 1994), the current version of Prograph is limited to producing applications – non-stand-alone code fragments like XCMDs, WDEFs, PhotoShop plug-ins, etc. can't be implemented with Prograph, although Prograph International plans to remove this restriction. In addition, Prograph only builds 68K-based Macintosh applications, although here too, they have plans to produce PowerPC, Windows, and Unix applications.

This article will describe the Prograph language and environment, and then discuss the Prograph application framework, the ABCs, in some detail. Where appropriate, comparisons will be made with other Macintosh application frameworks, most notably MacApp.

PROGRAPH – THE LANGUAGE

The Prograph language is an iconic language. The programmer codes by constructing drawings and the Prograph interpreter and compiler execute those drawings. Figure 1

shows a simple piece of Prograph code which sorts, possibly in parallel, three database indices and updates the database. To many programmers, drawing a program seems very strange at first, but the power and clarity of code expressed in Prograph is undeniable. The power of the language has to do with its primitives and the ways in which they can be used to express a computation, and explaining the language's syntax and semantics is the purpose of this section. The clarity of code expressed in Prograph is due, in part, to the fact that algorithms, by their very nature, have an inherent internal structure. Often, implementing an algorithm in a textual, and thus necessarily linear fashion, hides this structure. In contrast, the Prograph language brings this structure to the surface and enables the Prograph-literate programmer to grasp this structure in a single gestalt.

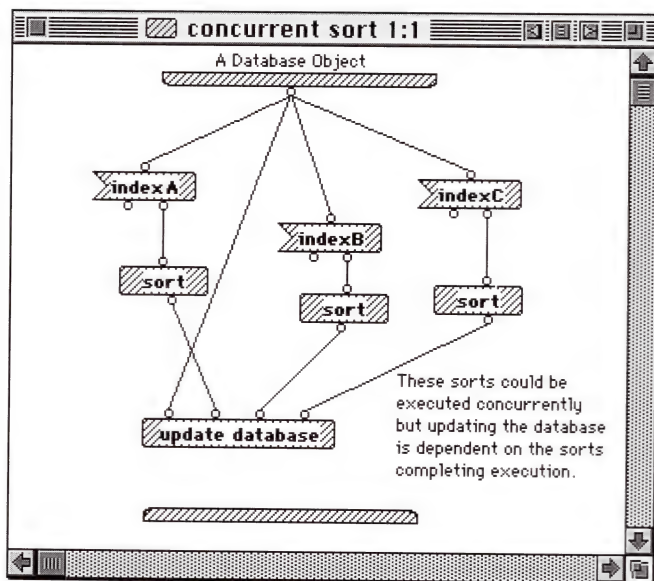


Figure 1. A simple piece of Prograph code which sorts –possibly in parallel– three indices and updates the database. Data flows from top to bottom, and operations – represented by icons – can execute whenever all their inputs are satisfied.

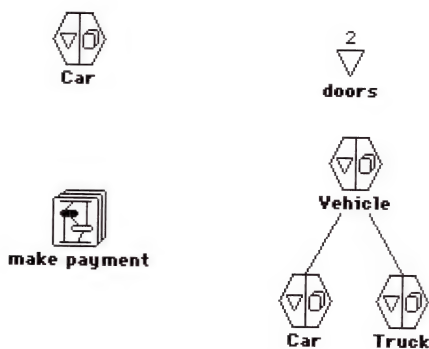


Figure 2. Some of the basic elements of the graphical design 'language'. Classes are represented by hexagons, data elements by triangles, pieces of code by

rectangles with data flow code inside, and inheritance by lines or downward pointing arrows.

The Prograph language has a graphical design and consistency which can be expressed in a small number of basic representations. Among these representations are: classes are represented by hexagons, data elements by triangles, pieces of code by rectangles with a small picture of data flow code inside, and inheritance by lines or downward pointing arrows. We see in Figure 2 some of these representations used in their simplest forms. The representations can then be combined and used in a variety of language elements. For example, initialization methods – which are associated with an individual class – are depicted as hexagonally shaped icons with a small picture of data flow code inside. Triangles represent instance variables in a class' Data Window to show that they're data. Hexagons drawn with edges and interiors similar to the instance variable triangles represent class variables. This associates them with the class as a whole while also associating them with data. Figure 3 shows some of these more complex representations.

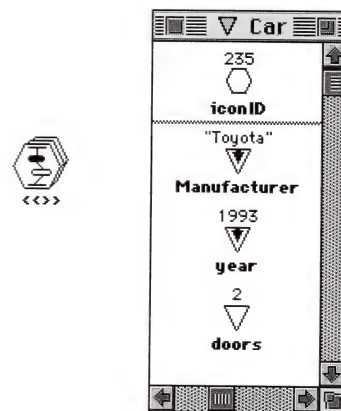


Figure 3. The elements of the graphical design 'language' representations of Prograph can be combined in ways that are surprisingly self-consistent and easy to learn. In this figure we see three examples of this: initialization methods, instance variables, and class variables.

Object-Oriented

The Prograph language is object-oriented. In particular, it is a class-based, single inheritance (a subclass can only inherit from one parent), object-oriented language with dynamic typing and a garbage collection mechanism based on reference counting. The programmer can design and implement new classes, visually specifying the new class' inheritance, additional instance or class variables, modified default values for inherited instance or class variables, and new or overridden methods.

Polymorphism allows each object to respond to a method call with its own method appropriate to its class. Binding of a polymorphic operation to a method in a particular class happens at run-time, and depends upon the class of the object. The syntax for this 'message send' is one of the more unusual aspects of Prograph's syntax. The concept of 'message sending' per se is not used in Prograph. Rather, the idea of an annotation on the name of an operation is used. There are four cases:

- No annotation means that the operation is not polymorphic.
- '/' denotes that the operation is polymorphic and is to be resolved by method lookup starting with the class of the

object flowing into the operation on its first input.

- ‘//’ denotes that the operation is polymorphic and is to be resolved by method lookup starting with the class in which this method is defined.
- ‘//’ plus super annotation means that the operation is polymorphic and resolves by method lookup starting with the superclass of the class in which this method is defined.

In addition, Prograph is one of the few object-oriented languages in which there is no SELF construct (or a ‘this’ construct, to use the C++ terminology).

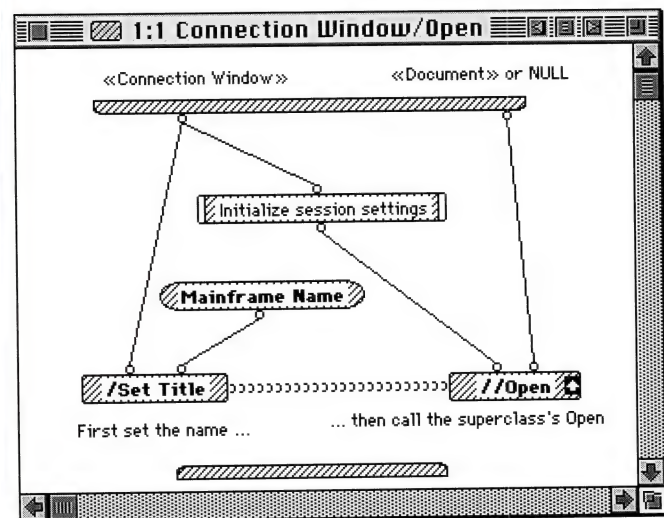


Figure 4. A typical piece of Prograph code showing comments, use of different types of operations, data flow (straight lines) and synchronization primitives (the wavy line).

Prograph has an almost completely iconic syntax, and this iconic syntax is the only representation of Prograph code. (Figure 4 shows a small code fragment typical of the Prograph language.) There is no textual equivalent of a piece of a Prograph program – the Prograph interpreter and compiler translate directly from this graphical syntax into 68K code. The icons of the Prograph language represent some twenty types of operations and Figure 5 shows most of the lexicon of the language. The inputs and outputs to an operation are specified by small circles at the top and bottom, respectively, of the icons. The number of inputs and outputs – the operation’s arity – is enforced only at run-time. In addition, there are a variety of annotations that can be applied to the inputs and outputs, or to the operation as a whole to implement looping or control flow. The visual syntax of Prograph has been formally specified. [Cox and Pietrzykowski, 1988]

Visual languages like Prograph sometimes suffer from the spaghetti code problem: there are so many lines running all over that the code for any meaningful piece of work actually ends up looking like spaghetti. Prograph deals with this issue by enabling the programmer to ‘iconify’ any portion of any piece of code at any time during the development process. This iconified code is called a ‘local’. Effectively, locals are nested pieces of code. There is no limit to the level of nesting

possible with locals, and there is no execution penalty to their use. In addition, locals can be named and this naming, if done well, can provide a useful documentation for the code. Figure 6 shows the same piece of Prograph code with and without the use of locals. Note also that long names, often containing punctuation or other special characters can be used for the names of locals. The proper use of locals can dramatically improve the comprehensibility of a piece of code. The one negative aspect of their use is the so-called “rat hole” phenomena – every piece of code in its own rat hole. This can sometimes make finding code difficult.

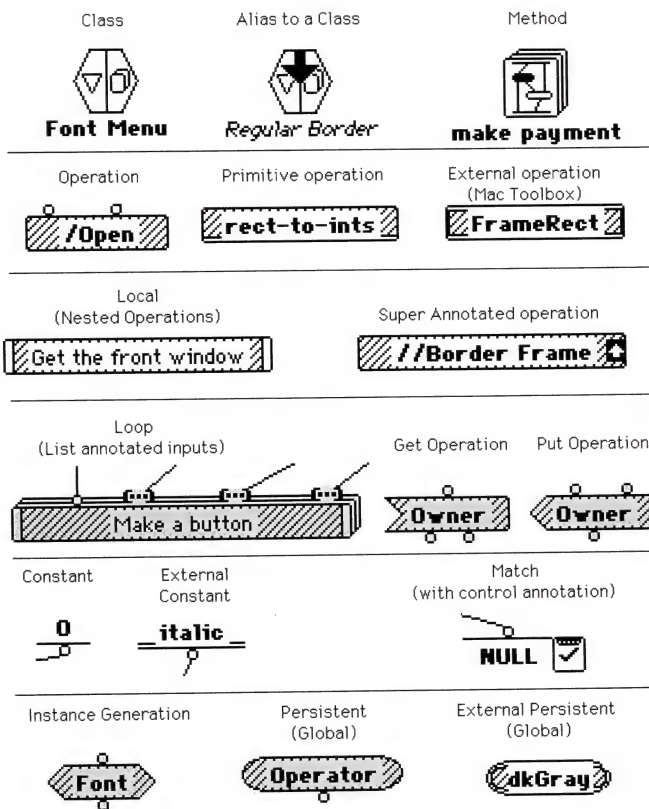


Figure 5. Most of the iconic syntax of the language.

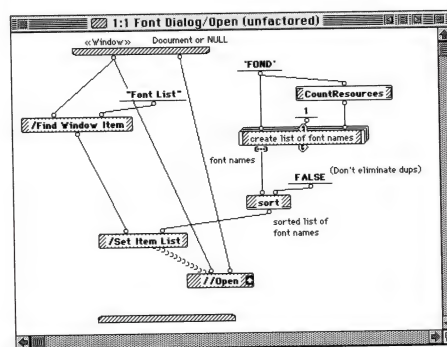


Figure 6a. The code to build a dialog box containing a scrolling list of the installed fonts without the use of Prograph locals to factor the code.

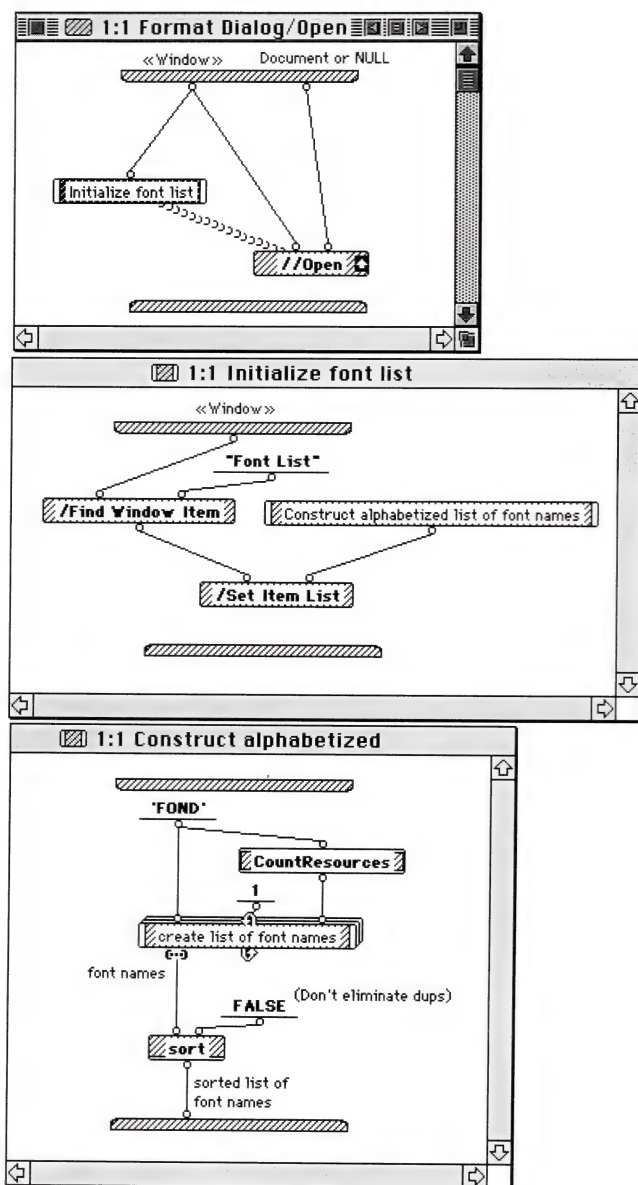
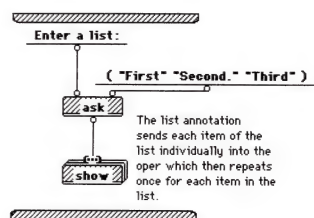


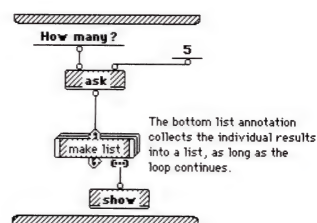
Figure 6b. The same code with Prograph locals to factor the code.

Three of the most interesting aspects of the Prograph syntax are list annotation, injects, and control annotation. Any elementary operation can be made to loop by list annotating any of its inputs. When this is done, the operation is invoked multiple times – one time for each element of the list that is passed on this input. Thus, list annotation on an input causes the compiler to construct a loop for the programmer. Since this annotation can be made on a local as well as a primitive operation, this looping construct is quite powerful. In addition, list annotation can be done on an output. On an output terminal, list annotation causes the system to gather together all the outputs at this terminal for every iteration of the operation and to pass as the 'final' output of the terminal this collection of

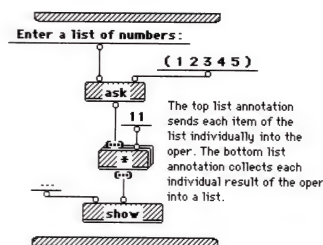
results as an output list. List annotation, then, enables the programmer to easily make an operation into a loop that either breaks down a list into its component elements and runs that operation on the elements, or to builds up a list from the multiple executions of an operation. An individual operation can have any number of its inputs or outputs with list annotations. Figure 7 shows several examples of list annotation.



Example 1



Example 2



Example 3

Figure 7. Examples of list annotation, an efficient mechanism for iterating over a list. The operation will automatically be called repeatedly for each element of a list, or its outputs will be packaged together into a list.

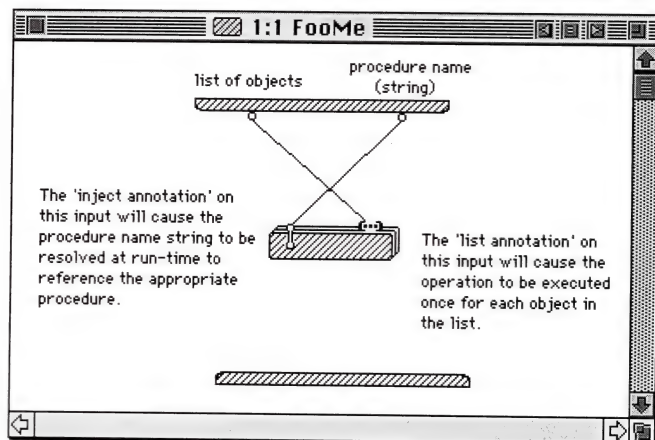


Figure 8. Simple use of inject to determine a procedure at run-time.

Inject lets you pass a name at run-time for an input which expects a function. For those familiar with Smalltalk, Prograph's inject is similar to Smalltalk's perform. It is also not too dissimilar from procedure variables in Pascal or function pointers in C. Suppose, for example, that you want to implement a function *FooMe* that takes two arguments: a list of objects, and a reference to a method to be applied to each of those objects. In Object Pascal pseudo-code, this function might look something like this:

```
Function FooMe(theList: TList; Procedure DoThis(object: TObject));
{ Iterates through the list of objects applying the DoThis procedure }
BEGIN
    theList.ForEach(DoThis);
END;
```

The Prograph implementation of *FooMe* would look something like Figure 8. Note that just the name – as a string – of the method to be applied to each object is passed to *FooMe*. This string is turned into a method invocation by the inject.

The Prograph representation of inject – a nameless operation with one terminal descending into the operation's icon – is a particularly well-designed graphic representation for this programming concept. When properly used, inject can result in extremely powerful and compact Prograph implementations of complex functions. However, when used improperly, it can result in code that is very difficult to read or debug, not unlike the computed GOTO of FORTRAN.

Control flow is the most unusual aspect of the Prograph language and is perhaps the most difficult aspect of the Prograph language for the inexperienced Prographer. It is a combination of annotations on operations and a case structure. A Prograph method can consist of a number of mutually exclusive cases. These cases are somewhat similar to the cases in a Pascal CASE statement or a C switch expression. The main difference is that unlike Pascal or C, the code to determine which case will be executed is inside the case, not outside it. A small example may make this clear. Suppose that we want to

implement a small function that has one integer input, and if that input is between 1 and 10, the value of the function is computed one way; if it is between 11 and 100, another way, and if it is anything else, yet a third way. In Pascal-like pseudo-code, this function would look something like this:

```
Function Foo( i: Integer): Integer;
Begin
    Case i of
        1..10:  Foo := Calculation_A(i);
        11..100: Foo := Calculation_B(i);
        Otherwise Foo := Calculation_C(i);
    End; {Case}
End; {Foo}
```

The implementation of *Foo* in Prograph would also involve three cases, and it would look like Figure 9. The control annotations are the small boxes on the right of the match primitives at the top of the first two cases. (Note that the window titles show the total number of cases in a method, and which case this window is, as in "2:3" the second of three cases.) In this simple example, the same annotation – a check mark in an otherwise empty rectangle – is used. The semantics of this annotation are: "If this operation succeeds, then go to the next case." The check mark is the iconic representation of success, and the empty rectangle represents the 'go to the next case' notion. If the check mark were replaced by an 'x', then the semantics would have been: "If this operation fails, then go to the next case". In addition to the otherwise empty rectangle, there are four other possibilities, and the five different semantics of control annotations are shown together in Figure 10. It is a run-time error and a compile-time warning to have a next-case control annotation in a location where it cannot execute, for example, in the last case of a method.

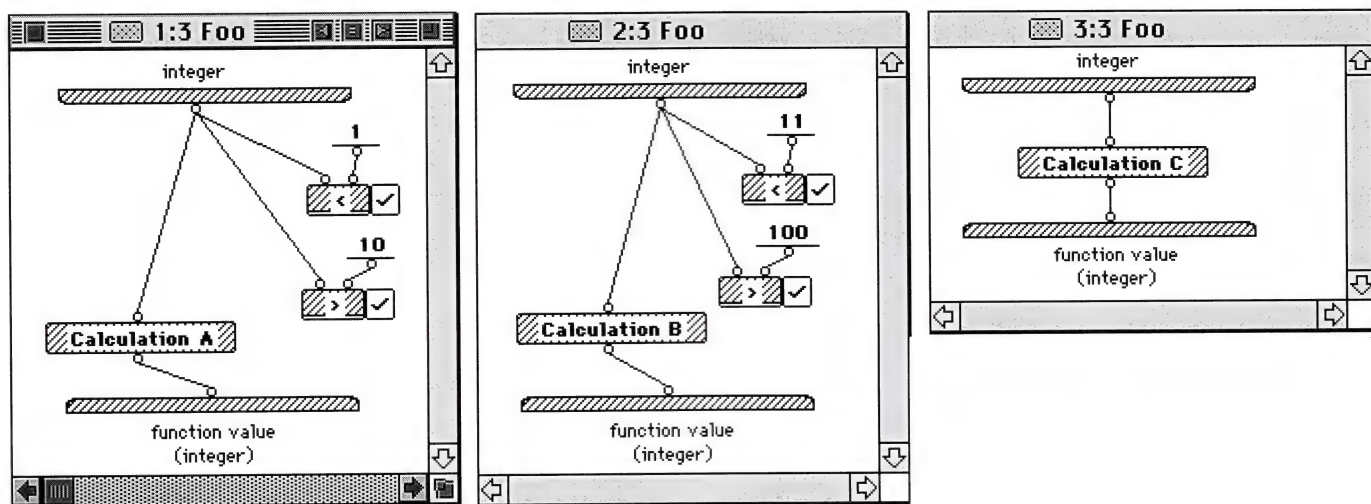


Figure 9. The Prograph equivalent of a small Pascal Case statement.

	Next case on success
	Next case on failure
	Terminate on failure
	Continue on failure
	Finish on failure
	Propagate failure on failure

Figure 10. The different actions possible with a control annotation.

DataFlow

The Prograph language is also a data flow language. Typed data elements flow along the data wires from the outputs of one operation to the inputs of another. Execution of an operation can take place any time that data is available on all of its inputs, but otherwise the order of execution is non-deterministic. This is often a conceptual stumbling block for the new Prograph programmer, probably because of prior training in linear, textual languages with their implicit execution order. In reality, however, order of execution is often an unnecessary detail in the implementation of an algorithm and Prograph frees the programmer from having to specify this unneeded detail – once, of course, that the programmer learns to differentiate the unnecessary specification of ordering from that which is necessary. Should an algorithm require that a particular operation, named A, be executed prior to another operation, B, the programmer can enforce this by connecting a synchro from A to B and Prograph will guarantee that A is executed before, although not necessarily immediately before, B. (An example of a synchro in use can be seen in Figure 4. This synchro ensures that the window's title is set before the superclass's Open method is invoked.) As you would probably expect, it's not uncommon to see beginning Prograph programmers use a few unnecessary synchro connections.

Data Types and Primitives

There are ten data types in the Prograph language: boolean, integer, real, string, list, external structure, NULL, NONE, undefined, and object. Type checking is performed at run-time at the execution of each operation. A type error will cause the application to halt. If the application is being run under the interpreter when a type error occurs, then control will be transferred to the interpreter and the programmer will be given the opportunity to correct the error – either by modifying the code or by modifying the data value – and continuing execution. If the error occurs in a compiled application, the application will catch the error, quit, and return to Finder.

In addition, there are 307 primitives that are effectively part of the language. These primitives include functions in the

following areas, among others: math, list processing, I/O, string processing, and interpreter control. Figure 11 presents the entire set of primitives.

New external primitives can easily be added. In fact, this is how the Mac Toolbox traps are supported: as 'external' primitives defined in files that are supplied with CPX. It is relatively easy to add new external primitives, and this is the means that existing libraries of C or Pascal functionality can be brought into Prograph. Once this is done from some library, these external primitives are supported at exactly the same level as Mac Toolbox traps. Thus, it is not the case that C or Pascal libraries that you bring into Prograph are somehow 'second-class' citizens in the Prograph language. This is the means by which new Mac system calls (e.g., QuickTime or the Drag Manager) are added to Prograph. Prograph International tries to keep up with the steady stream of additional Mac APIs coming out of Apple, but if for some reason they have not yet gotten to the nifty new API that you want to use, you can just do the work yourself to bring this functionality into Prograph as a new external primitive. There is no easy way today, however, to bring in C++ libraries, especially in a way that would retain the inheritance structure of the C++ library.

PROGRAPH – THE ENVIRONMENT

In this section a quick (and necessarily incomplete) overview of the Prograph programming environment will be presented. Normally the programming environment, although of great importance to the actual use of the language and framework, is tangential to a discussion of the syntax of the language and the design of the framework. For Prograph this is not true, due to the highly integrated nature of all three of these. However, to keep this chapter of manageable length and because the main subject of this book is frameworks, this subsection on the Prograph environment will be brief.

The Prograph environment consists of two tools: a compiler and a combined editor/interpreter/debugger, usually called the interpreter or the just simply 'the environment'. (Calling this second tool the 'interpreter' is an historical artifact that is now technically inaccurate since the current version of the 'interpreter' is an incremental compiler.) Typically, an application is designed, implemented, and debugged in the environment and then as a final step, compiled by the compiler. In this section, each of the three aspects of the environment will be described separately. This separation is a convenience for the purposes of exposition.

Editor/Interpreter/Debugger

The editor is a structured graphics editor that understands the syntax of the Prograph language. This editor simultaneously assists the user in entering Prograph code with a minimum of hassle and enforces Prograph syntax so that it is not possible to enter syntactically incorrect Prograph expressions. The editor also enters the correct arity for any primitive, universal, or method after the programmer enters the name.

ABC Support

draw-style-text
popup-menu

AppleTalk

ATP-Close
ATP-Get-Request
ATP-Get-Response
ATP-Open
ATP-Send-Request
ATP-Send-Response
NBP-Close
NBP-Confirm
NBP-Lookup
NBP-Open
NBP-Register

Bit Manipulation

bit-and
bit-not
bit-or
bit-shift-l
bit-shift-r
bit-xor
test-all?
test-bit?
test-one?

Callbacks

callback
dispose-callback

Data

copy
inst-to-list
list-to-inst
shallow-copy

DataFile Manager

cluster-delete
cluster-lock
cluster-read
cluster-replace
cluster-unlock
cluster-write
db-backup
db-close
db-compact
db-delete
db-flush
db-info
db-list
db-new
db-open
db-rename
db-shutdown
db-wait
key-close
key-delete
key-find
key-first
key-info
key-last
key-list
key-new
key-next
key-open
key-previous
key-read
key-rename
key-value
table-close

table-delete
table-export
table-import
table-info
table-list
table-new
table-open
table-rename

Environment

gestalt
gestalt-attribute?
trap?

File

close
create
delete
file-size
get-file
get-position
open
put-file
read
read-line
rename
resource-file
set-position
write
write-line

Graphics

find-bounds
ints-to-point
ints-to-rect
ints-to-rgb
point-to-ints
point-to-rect
rect-to-ints
rect-to-points
rgb-to-ints

Input/Output

accept
answer
answer-v
ask
display
print-text
select
set-dialog-font
show

Interpreter Control

abort
abort-callback
call
compiled?
debug
execute
find-method
halt
open-info-window
open-method-window
switch-to-prograph
trace
yield-cpu

Lists

(in)
(join)
(length)
attach-l

attach-r
detach-l
detach-nth
detach-r
find-instance
find-sorted
get-nth
insert-nth
make-list
pack
reverse
set-nth
set-nth!
sort
split-nth
unpack

Load & Save/Data Clustering

clear-bytes-map
from-bytes
load
save
to-bytes

Logical/Relational

<
<=
=
>
>=
and
choose
not
or
switch
xor
~=

Math

*
**
+
++
+1
-
—
-1
abs
acos
annuity
asin
atan
atan2
compound
cos
div
exp
idiv
ln
log10
max
min
pi
power
rand
round

round-down
round-up
set-seed
sign
sign-extend
sin
sqrt
tan
trunc
÷
÷÷

Memory

address-to-object
block-address
block-size
compact-memory
from-handle
from-pointer
get-integer
get-point
get-real
get-rect
get-string
get-text
heap-size
lock-block
lock-string
make-direct
make-handle
make-pointer
memory-callback
new-block
object-to-address
put-integer
put-point
put-real
put-rect
put-string
put-text
string-address
to-handle
to-pointer
unlock-block
unlock-string
valid-heap?

Serial Port

break-serial-port
clear-serial-port
configure-sport
count-sport-input
get-sport-buffer
get-sport-refs
kill-serial-port
open-serial-port
receive-serial-port
send-serial-port
send-sport-done
set-sport-buffer
sport-configuration

String

"in"
"join"
"length"
format
from-ascii
from-string
integer-to-string

middle
prefix
string-to-integer
suffix
to-ascii
to-string
tokenize

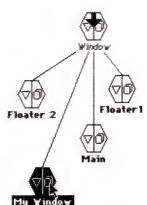
System

ancestors
attribute-com
attributes
called-from-get
called-from-meth
calls-to-get
calls-to-meth
calls-to-set
children
class-com
class-section
classes
create-class
create-method
descendants
editor-methods
meth-com
meth-com-g
meth-com-s
meth-io-com
meth-io-com-g
meth-io-com-s
method-arity
method-classes
method-section
methods
pers-com
persistents-section
persistents
section-com
section-content
sections
set-attribute-com
set-class-com
set-meth-com
set-meth-com-g
set-meth-com-s
set-pers-com
set-section-com

Type

boolean?
external-type
instance?
integer?
list?
number?
real?
string?
type

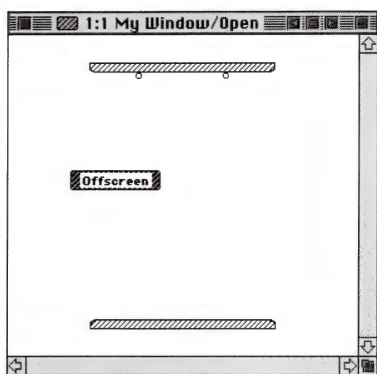
Figure 11. A list of the Prograph primitives organized by category.



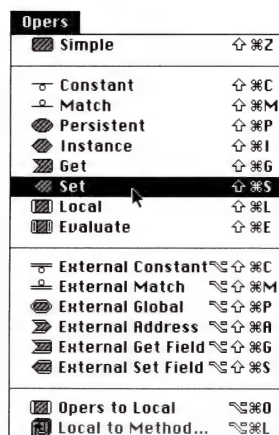
Step 1: Double click on the right side of the "My Window" class icon. This will open this class's methods window.



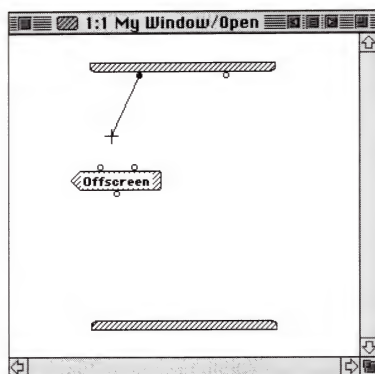
Step 2: Click in the methods window to create a method. Type "Open" for its name.



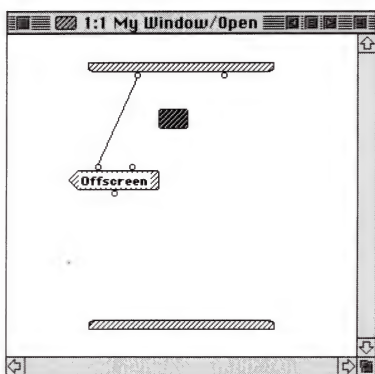
Step 3: Click twice just beneath the input bar in order to create two inputs, since the arity of Window/Open is two inputs and no outputs. Click to create a new operation, and name it "Offscreen".



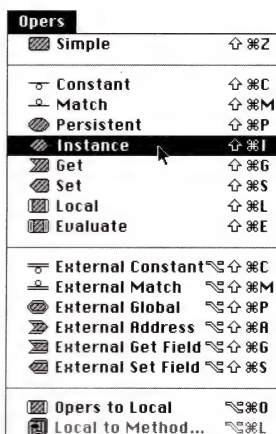
Step 4: Choose the Set item on the Opers menu to change the simple operation "Offscreen" into a Set operation.



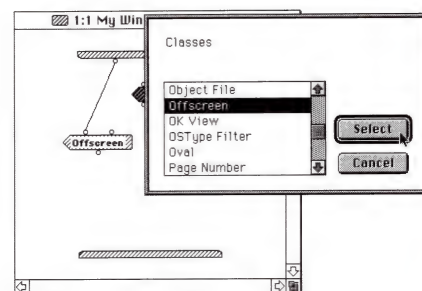
Step 5: Select the left input, and then hold down the Option key. Move the mouse to the left input of the Get operation to draw a flow line.



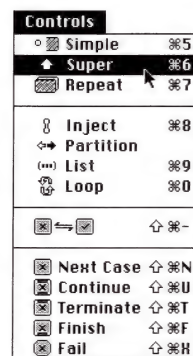
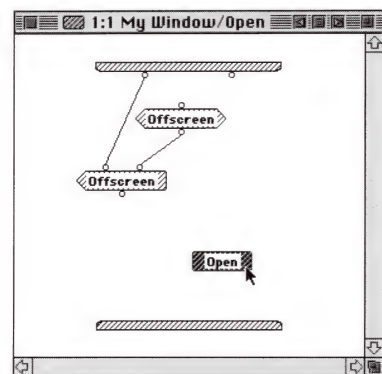
Step 6: Click to create a new operation.



Step 7: Choose the Instance item on the Opers menu to change the simple operation into an Instance Generator.



Step 8: Double-click on the right side of the Instance Generator, and the select "Offscreen" from the scrolling list of class names.



Step 9: Click to create a new simple operation and name it "Open".

Step 10: Choose the Super item from the Controls menu to add the Super annotation to the simple "Open" operation.

Step 11: The super annotation, plus the "//", plus the comments from the overridden method are added to the operation.

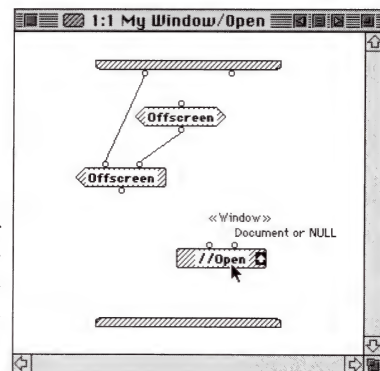
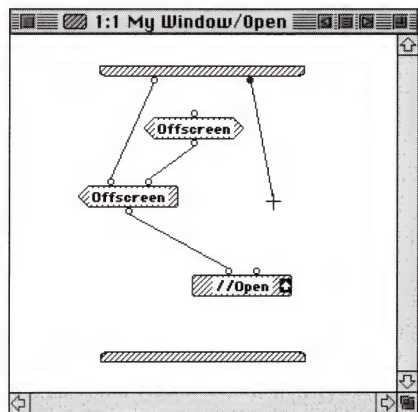
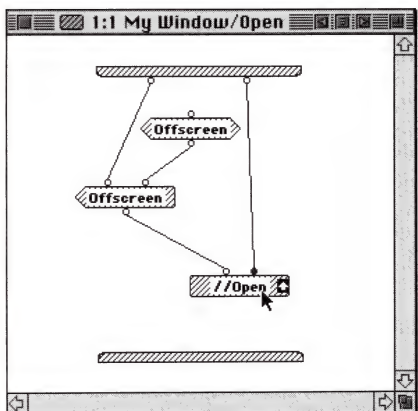


Figure 12. (above and right) The Prograph editor provides an ideal environment for entering, testing, and debugging Prograph code.

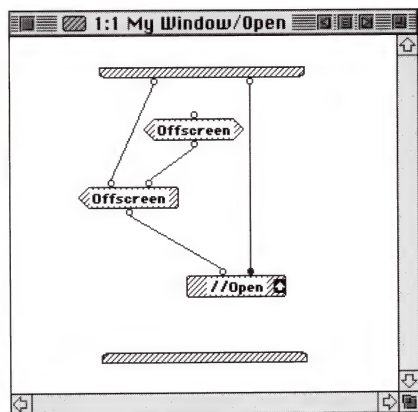
This long figure documents the series of actions that a programmer would take to enter a small method. Note that because of the support of the Prograph editor, it is not possible to enter a syntactically incorrect Prograph expression. In effect, the Prograph editor is the graphical Prograph language what a syntax-directed editor can be to a textual language.



Step 12: Command-click on the inputs to the super annotated Open, in order to hide the comments. Then connect the inputs to this operation by clicking, holding down the option key, and moving the mouse. Do this for each of the two inputs.



Step 13: The method is now complete, but in order to 'straighten' things up a bit, click on the second input to the super annotated Open, and press this space bar. This will make the flow line vertical.



Step 14: The method is now finished.

(There is also a mechanism so that the name does not have to be typed in.) Figure 12 shows the sequence of editor actions for entering a simple Prograph method.

The environment includes an incremental compiler and you can run the application you're developing under control of the environment. While doing so, you can interrupt the application, examine data values (Figure 13), change data values and code, and resume execution of the application.

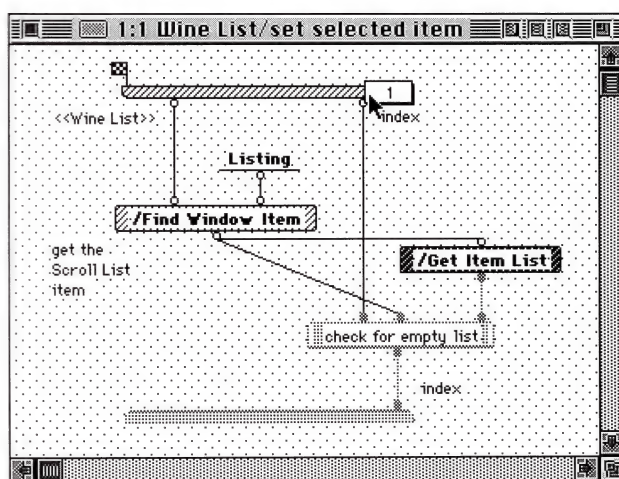
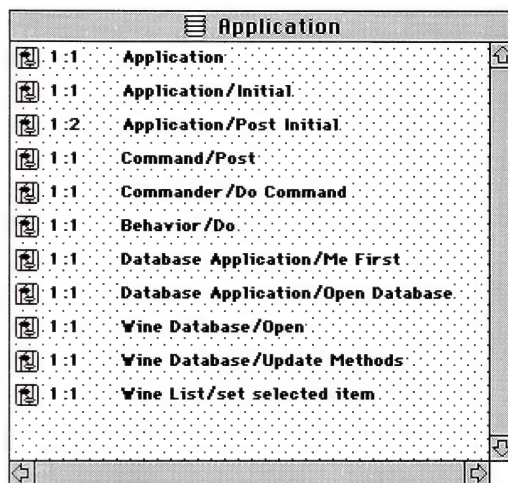


Figure 13. Inspecting data values in code when execution is interrupted at a breakpoint. Execution can be resumed, or can be carried out operation by operation (analogous to line-by-line execution in debuggers of textual languages).

In effect, the environment includes a graphical interface builder, but implemented in a somewhat unusual manner. Specific editor classes – themselves written in Prograph – provide individualized graphical editors for each one of the main ABC classes. Figure 14 shows a collection of these editors. These editor classes – called the ABEs – are executed in separate lightweight processes when the programmer is graphically editing one of the ABC instances. The ABEs are present only in the environment, and are not included in a compiled application. For this reason, they are also called interpreter-only classes or sections.

Compiler

After an application has been developed and debugged, it can be compiled in what is essentially a batch process. The resulting application is a standard, double-clickable Macintosh application. In a later section, comparisons of the RAM footprint

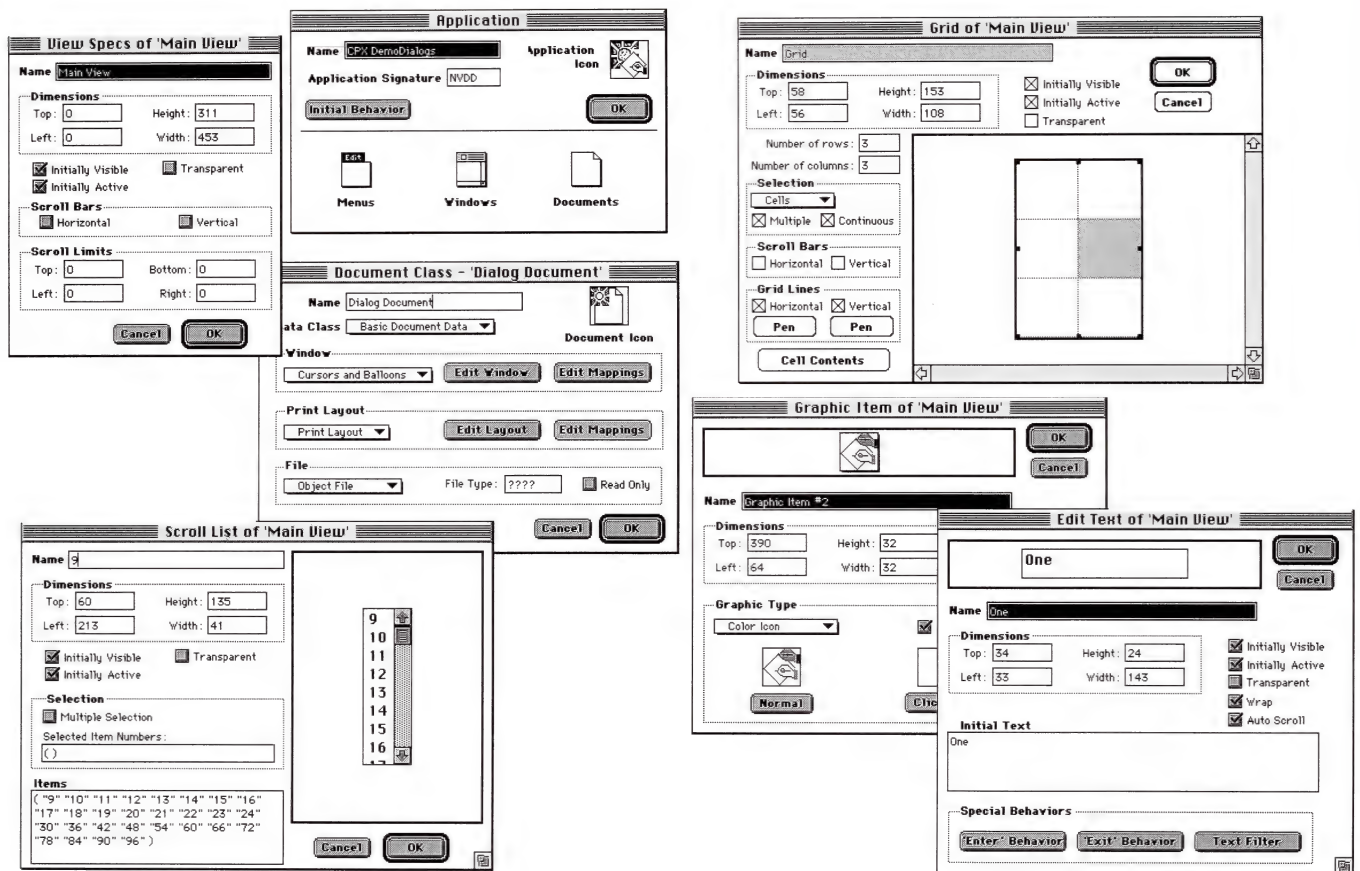


Figure 14. A collection of Application Builder Editors — graphical editors for many of the classes in the application framework.

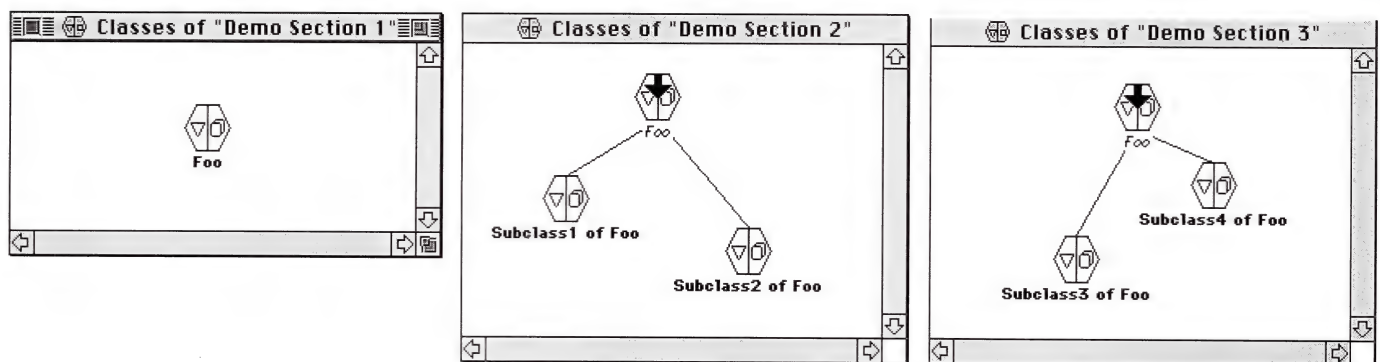


Figure 15. The use of aliases to classes in different sections.

Note that the iconic representation for the alias — a class icon with a super-imposed downward pointing arrow and an italic name — is the nice mix of the graphical language of the Macintosh Finder and that of the Prograph language.

and execution speeds for compiled applications will be made.

Other features of the Environment

There are many more aspects of the Prograph CPX environment than can be covered in this short chapter, including a help system for all the Prograph primitives and ABC classes and methods, as well as all the Mac Toolbox calls, a variety of accelerators for 'power' users, the manner in which comments and other documentation are supported by the environment, and the way in which the environment is actually used in practice to support an 'outside-in' development style.

PROGRAPH – THE APPLICATION FRAMEWORK

In this section, basic statistics and run-time architecture of the Prograph ABCs are explained. Particular attention is paid to the run-time inter-connections between the objects that make up a running application, as opposed to the static inheritance structure of the class library. Since both the ABCs and the Prograph language itself are tightly integrated with the development environment, a lengthy explanation of how the programmer uses the ABCs in the Prograph environment is given. This section concludes with some measurements of RAM and disk footprints of several applications written with the ABCs and compares these to the equivalent MacApp applications.

Basic Statistics and Static Structure

The 161 classes of the Prograph ABCs are structured as 46 sections – independent units of compilation. Each can have classes as well as global functions and global data. References to the sections that comprise a complete application are stored as a project. (The project stores little else besides references to sections and Macintosh resources (icons, sounds, cursors, and other Macintosh Toolbox structures.)) Names of classes, global functions, and global data must be unique in a project. The programmer can add any number of new sections to a project.

Inheritance links can span any number of sections. The programmer's interface for this is a particularly elegant extension of the notion of an alias from the Macintosh Finder [Apple, 1993]. While there can be only one class named 'Foo' in a project, there can be any number of aliases to this class in other sections. In all respects except adding new instance variables and changing the superclass, aliases act just like the original class. Thus, if there was a need to implement subclasses of 'Foo' in two different sections, the programmer would only have to add an alias to Foo in each section, and then implement subclasses of each alias. Figure 15 shows how this would look to the programmer. The representation of the alias follows both the graphical design of the Finder (names in *italic*) and that of the Prograph language (hexagons, downward pointing arrows).

The inheritance structure of the ABCs is that of a forest of a large number of short trees. There is no single "Object" class that is the superclass of all or most of the ABC classes. In fact, a large number of classes (approximately 60) inherit from no other class. Normally, this would be an unusual and perhaps

sub-optimal structure for an application framework. This is not the case for the ABCs because 'object' is a basic data type of the Prograph language and because of the large number of language primitives (approximately 300) many of which provide much of the functionality found in the Object class in other systems. For example, cloning, both shallow and deep, is accomplished via the copy primitive rather than a Copy method in class Object. This seems to work quite well, especially since even the built-in primitives like 'copy' can be overridden using the data-determined reference mechanism (where the class of incoming data determines which class will handle method calls). Figure 16 shows the inheritance structure of the ABCs.

Prograph CPX Class Forest



Figure 16. The inheritance structure of the class library.

There are 2327 methods in the 335 classes in the ABCs and ABEs, and all methods are provided in source code with the CPX product. Method sizes range from 0 operations (null methods) to more than 10 cases and 60 operations.

Run-time Architecture

The ABCs structure a running application in a way similar to other Mac frameworks. This isn't too surprising since they're all basically extensible object-oriented apps that implement the

same Macintosh user interface specification. The look and feel of that specification dramatically influences the functionality of the objects that make up any framework that supports it. Thus, most of the major objects of the Prograph framework will be familiar to users of these other frameworks:

Application – The root object that governs the application's behavior and visual appearance. It allocates and initializes most of the other objects that comprise the application's run-time structure.

Window – Object wrapper for a Macintosh window.

View – Rendering object.

Document – Defines a relationship between a window, a data file and a print layout. This object also provides the file and disk I/O behavior for the application. The actual data controlled by a document is stored in an auxiliary Document Data object.

Task – An object wrapper for request for an action, initiated by the user, or internally. (Not unlike the MacApp command object.)

Desktop – The root object that controls all the visual aspects of the application: windows, menubar, menus, palettes, etc. It is the root of the application's visual hierarchy.

Commander – The event and command handler for the application.

Menubar and **Menu** – Object wrappers for the menubar and menus.

adding new instance variables and adding and overriding methods as needed, extending the ABCs to reach your desired feature set. This often involves overriding null placeholder methods. For example, override View/Draw to let the framework get to your rendering code, and override Command/Undo so the framework can use your code to automatically handle undo processing. These null methods are sometimes called "hook" methods. [Schmucker, 1988]

Common tasks include:

- Subclass Application, override Application/Update Menus to handle any application-wide menus
- Subclass Window for each type of window desired.
- Subclass Menu for each menu that will be inserted in the menu bar. Add a method or universal to handle the menu event associated with each menu item.
- Subclass Task (or Command or Behavior) for each undoable action that the user can perform.
- Subclass Document and Document Data for each type of file that the application deals with. Override, at a minimum, Document Data/Get Data and Document/Put Data so that these files can be read and written.

(Notice that designing a custom View subclass for each rendering is not part of this list. More on this later.)

The environment relieves the programmer of the tedium of constructing the many Window and Menu subclasses that are needed for any significant application, as well as providing a great deal of flexibility in the design and implementation of hook methods.

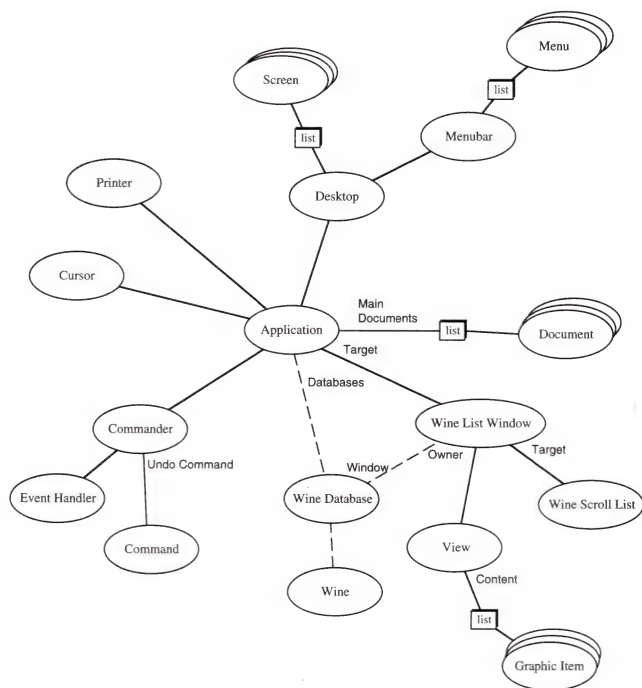


Figure 17. Run-time interconnections

Figure 17 shows the run-time interconnections between these objects, as they occur in a small database application that is shipped with CPX as a sample program.

Using the Framework

To build an application with the ABCs, as with any framework, you design and implement subclasses to the ABC classes,

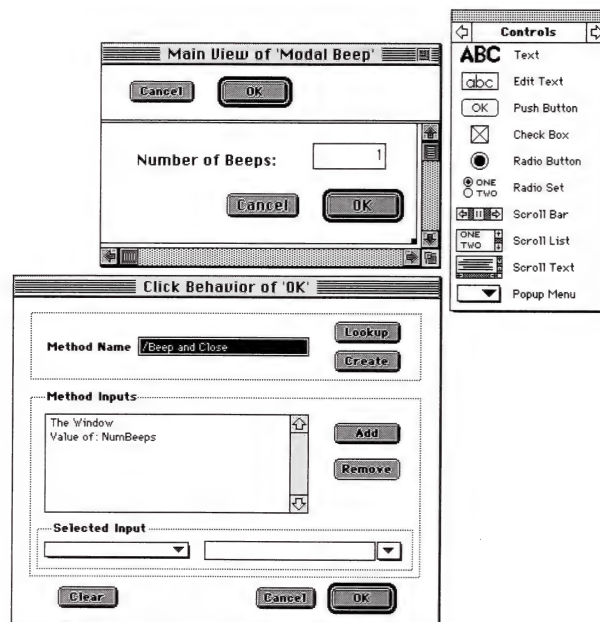


Figure 18. The Window and Button Click Behavior Editors for a simple window.

(Note that the widget palette – one of three – is also shown.)

Continued on page 82



By Scott T Boyd, Editor

OPENDOC GETS A SHOT IN THE ARM

Adobe Systems agreed to join CI Labs, the vendor-neutral association promoting OpenDoc. Adobe will provide financial support to the non-profit organization. In addition to incorporating OpenDoc technology into their applications, Adobe will also develop OpenDoc parts for their major video and graphics applications, allowing display and printing of documents in the content formats of Illustrator, Photoshop, and Premiere. Adobe's full sponsorship follows closely on the heels of Lotus Development's joining CI Labs in August.

GET THEE TO THE INTERNET

Kaleidospace, a commercial Internet site for the promotion and distribution of independent art is asking for contributions to online collaborative work by famous artists. Internet users can add to stories, graphic novels, music and even videos started by the celebrity artists. Kaleidospace also promotes, distributes, and sells works by independent artists, musicians, writers, performers, animators, filmmakers, CD-ROM authors and software developers.

Kaleidospace, <http://kspace.com>, PO Box 341556, Los Angeles, CA 90034. (310) 399-4349 voice, (310) 396-5489 fax, editors@kspace.com.

PPC CODE TRACER

Posted to macgifts@mac.archive.umich.edu, a first release of Peter J. Creath's Janus (0.1). Think of this as the PPC native equivalent of the ResEdit CODE editor. In layman's terms, Janus is a fat binary which lets you find the PowerPC native code corresponding to 680x0 code. Now you can patch your favorite native games to give yourself infinite ammo and lives too! NOTE: This is not for the faint of heart. If you can figure out the 680x0 patches on your own, this is for you. If not, find somebody who can.

Peter can be reached at pjcreath@phoenix.princeton.edu.

RECOMMENDED READING

The UNIX-Haters Handbook, by Simson Garfinkel, Daniel Weise & Steven Strassmann IDG Books, \$16.95, ISBN 1-56884-203-1 with foreword by Donald Norman, Apple Computer, and anti-foreword by Dennis Ritchie, AT&T. Comes recommended by a number of our readers.

PUT CONGRESS IN YOUR POCKET

Now available: the 103rd Congress, a book for the Newton. The 103rd Congress directory lists all members of

the United States House of Representatives and Senate. Included is the member's phone number, fax number, mailing address in Washington DC, and committee assignments.

\$25.00. Iverson Software Co., P. O. Box 3, Rice Lake WI 54868-0003. For more info, call Jeff Iverson at (715) 236-7918.

ADD SPELLCHECKING TO YOUR APP

SpellWright is a new royalty free spellchecking toolkit from LexTek Int'l. designed for applications which don't presently have spellchecking capabilities and as a replacement for spellcheckers which either have expensive licensing fees or are hampered by poor performance.

SpellWright supports five different languages: American English, British English, Dutch, French, and Spanish. Supplemental dictionaries cover Legal, Medical, Names, Pharmaceutical, Post Offices, Religion, and Science. SpellWright provides accurate suggestions, checks for capitalization errors, double word errors, or look up words by pattern.

For more information, contact LexTek International at 2255 N. University Parkway, Suite 15, Provo, UT 84604. (801) 375-8332 voice, (801) 377-7654 fax.

NEW SCRIPTGEN

StepUp Software today announced that it is shipping ScriptGen Pro 2.0, which adds support for Apple's new Installer 4.0, boasts speed increases over 400%, and adds developer hooks for customization. Installer action atoms, setup functions, and rules can be 'plugged' into any script.

ScriptGen Pro 2.0 supports the Installer's new 'application folder interface' which allows users to select where application files are installed. Hierarchical views of custom packages are now supported. Easy Install rules - including System 6 versus System 7 and gestalt tests - are new to version 2.0. Folders can be installed, eliminating the need to provide script information for every file included in the installation.

ScriptGen Pro version 2.0, reengineered for speed, generates scripts greater than 400% faster than previous versions. Typical 2.0 documents require less than one-third the disk space of earlier versions. ScriptGen Pro supports decompression of Stuffit, Diamond, and Compact Pro format archives, with additional software.

\$169. Upgrades \$49 Canada/US, \$59 elsewhere. Free demo is available on AppleLink (Third Party Demos:Developer Tools), CompuServe (MACDEV), America Online (MDV), and e-World (Straight to the Source:Info Samples:StepUp Software).



The Window and Menu subclasses are typically implemented indirectly via the graphical interfaces of the ABEs. Using the Window editor, for example, the programmer can not only design the graphical layout of the window instances, but can also specify the window's resize properties (whether its views grow or shrink as the window's size changes, for example.), the draw behavior and the key handling for its views, and the click behavior for its buttons. This is all done in a Behavior Editor. (Figure 18 shows the Window Editor and the associated Behavior Editor for one of its buttons.) When you want to change the way a class acts, you write methods for it. On the other hand, when you want to express how a single instance should behave (such as a specific window or button), you use a behavior, a command that makes an interface element respond to an event.

Behaviors provide an elegant solution to a problem that many frameworks share: the rigidity of hook methods. Because most object-oriented languages, including Prograph, require that the interface of an overridden method be identical to that of the method it is overriding, the programmer is forced into using whatever the framework designer has specified. (Conversely, the framework designer is required to anticipate the needs of all the users of the framework when designing the interfaces to hook methods.) This can result in either an interface that does not pass all the necessary information, or to one that is overly complex for the simple or average cases. The ABCs avoid all of these problems by providing what is in essence a layer of indirection between the framework and the code that will be called. This layer of indirection is the Behavior Specifier object.

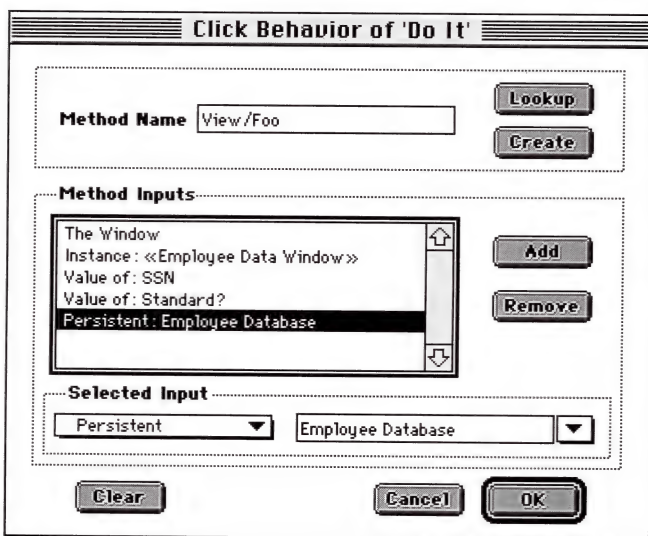


Figure 19. A Click Behavior Editor example where all the behavior for the button is obtained without writing any Prograph code.

A Behavior Specifier enables the programmer, in effect, to custom design the hook method for each object. Figure 19 specifies a click method for a button that will do the following:

- allocate an instance of the class Employee Data Window
- retrieves the current value from the editable text item named "SSN", and the boolean value of the checkbox named "Standard?"
- retrieves the value of the global (persistent) named "Employee Database".
- include these as arguments, together with the window that owns the button, in a call to the method View/Foo Button.

Achieving certain common tasks in the ABCs

To get a feel for the use of the ABCs, the following section sketches out the manner in which the ABCs are used to perform several common tasks in application development.

Activating a menu item

To activate a menu item you have to write some code – typically in an override of the Update Menu method present in many of the CPX classes – deciding programmatically what object controls the activation of this particular menu item, and under what circumstances it should be activated.

Menu items, for the purpose of enabling, are identified by the name of their associated menu behavior, an object that encapsulates the action that a menu item represents. Your code enables the menu behavior named "Foo" and the framework determines that this is, for example, the 4th item on the 3rd menu.

While the MacApp menu enabling is handled by a background process executing periodically, the menu enabling in CPX is done on demand when the user clicks in the menubar (or types a command key equivalent to a menu item). The Event Handler/Mouse Down method passes responsibility for handling mouse downs to the Desktop object. When the Desktop/Mouse Down method determines that the mouse down occurred in the menubar, it calls the Menubar/Mouse Down method. One of the first things this method does is update the enabling for all the menus.

Consider for example two menus named 'App Menu' and 'Win Menu'. The App Menu has three items and the Win Menu has two items. Let's consider the Win Menu first. Its two items, Win1 and Win2, are enabled by the windows Win1 and Win2. The code to do this, which is executed whenever one of these windows is the frontmost window, is shown in Figure 20. This code merely calls the inherited Update Menus method as well as executing a small local that uses the Set Item Enabled? method to enable the menu item whose behavior is named "Win1". (In this case the behavior, the menu item, and enabling object all happen to have the same name.)

All the items in the App Menu are enabled by the application object. This menu also shows that the menu item text can be different from their associated behaviors. The behaviors are named App1, App2, and App3 and these are listed in the application object's Update Menu code. However

the menu item text strings are App1, El App2, and Lé App3. The menu item strings can be localized to different natural languages or just changed in any way you require without affecting your code.

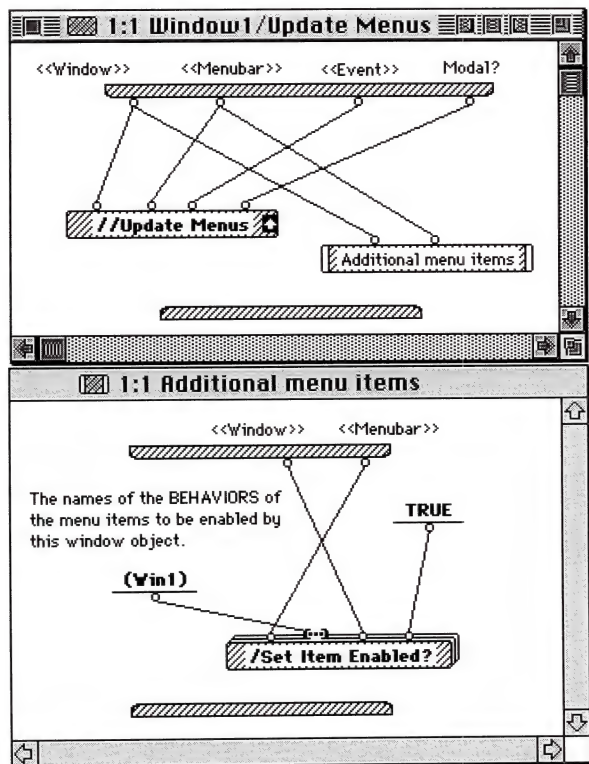


Figure 20. The menu enabling code for two menu items controlled by a window subclass. This is the simplest example of menu enabling.

Handling a menu event

In most cases you don't even have to write code to handle a menu event, but need only specify a Menu Item Behavior with a graphical editor which is part of the Menu class' ABE. This behavior can be constructed so that it calls the method or primitive of your choice, with whatever arguments are needed by that code.

Of course, whatever actions you want to take place as a result of the menu item being chosen will have to be coded, but the connection between the menu item and that code is done with a behavior.

Drawing the interior of a window

Surprisingly, you often don't need to write any code to connect your drawing code to a particular view; you connect it with a behavior. One advantage of this approach is that you typically don't need as many View subclasses as you would in a MacApp application. In effect, the connection between a particular view and its drawing (and also interaction behavior) is stored in its Drawing Behavior (and Click Behavior) instance variables rather than being done by subclassing View and overriding its Draw (and DoMouseDown) methods.

Measurements

Prograph, used in conjunction with the ABCs, produces small footprint, stand-alone applications. The minimal Macintosh-style "Hello, World" application – with multiple windows, the standard Apple, File, and Edit menus, and printing – has a RAM footprint of about 700K.

Figure 21 presents the results of a test to compare the RAM footprints of MacApp and Prograph CPX by re-implementing several of the MacApp sample programs in Prograph. These sample programs ranged in size from about 50 to about 3000 lines of C++. The RAM footprint of the Prograph version is generally about 300-400K larger than that of MacApp. About half of this difference is attributable to the overhead of the additional symbolic data necessary to support the inject feature of the Prograph language, and a portion of the remainder can be attributed to the Prograph garbage collector.

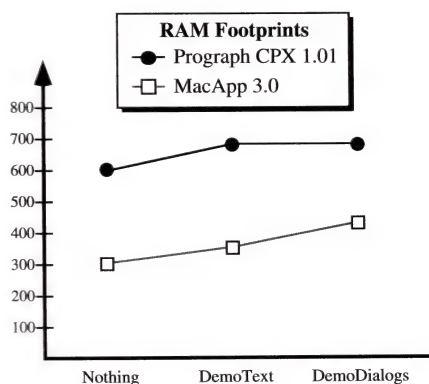


Figure 21. RAM footprint comparisons (in K) for the re-implementation of three MacApp samples in Prograph CPX.

CONCLUSION

The Prograph application framework is comparable in its breadth and depth to other commercially supported frameworks, but when the power of the Prograph language and the speed of its development environment are taken into account, Prograph CPX stands out as a superb application development tool.

REFERENCES

- [Cox and Pietrzykowski, 1988] P.T. Cox and T. Pietrzykowski, "Using a Pictorial Representation to combine DataFlow and Object-orientation in a language independent programming mechanism", Proceedings of the International Computer Science Conference, 1988, pp. 695-704
- [Parker, 1993] Parker, Richard O., *Easy Object Programming for the Macintosh using AppMaker and THINK Pascal*, Prentice-Hall, 1993
- [Schmucker, 1988] Schmucker, Kurt, *Object-Oriented Programming for the Macintosh*, Hayden, 1988
- [Schmucker, 1994] "DemoDialogs in Prograph CPX", *FrameWorks*, Volume 8, Number 2, (March/April 1994), pp. 8-13.
- [TGS, 1989] TGS Systems, "Prograph Syntax and Semantics", Prograph 2.5 manuals, Appendix IV, Sept. 1989 (first printing), July 1990 (second printing)





By Scott T Boyd, Editor

DON'T SCREAM, SEND 'EM THE ARTICLE!

I'd like to applaud Eric Shapiro's article entitled "Multiple Monitors vs. Your Application". As a sometimes one, two, or three monitor user I routinely experience everything he griped about. Purchasing multiple monitors is the most cost effective solution for attaining a large on-screen workspace and developers would do well to support it better. (Having multiple monitors has also worked in the past to really freak out some of my IBM user/programmer friends!)

Even Apple has problems writing friendly windowing code. I can't tell you how many times I've disconnected my second monitor to use somewhere else, restarted, launched AppleTalk Remote Access 1.0, and found one of its crucial windows to appear offscreen. I have to reconnect the second monitor, move the windows to the main monitor, and try again! At least when I encounter this bug in Excel I can "arrange" it back on...

I too was aghast to see that the "new version of one popular developer tool" creates the window and then snaps it into position. The first few times I brought up a window I was sure that two were appearing. How could something that obvious slip through, especially when the fix should be a single line of code? Anyway, thanks for the article. If I see one more window jump back to my main monitor when I try to enlarge it on my second monitor by clicking the zoom box, I'll scream!

— Jeff Mallett, j.mallett@genie.geis.com

PPC ASSEMBLY ARTICLE COMMENTS

When Bill Karsh repeated last month the worn-out advice originally promoted by the Apple folks last year ("You don't need assembler, the compiler can do better than handwritten assembly" or words to that effect), it hit me with particular irony. You see, the lack of adequate compiler tools (Thanks, Apple, for your inimitable support here) has forced me to write more assembly code for the 601 in the last couple months than for all other computers combined over the previous decade. Anyway I read his article with great interest. Some comments:

1. Perhaps your readers should know that the sequence,

```
addis r3,0,0xABCD
addi r3,0,0xEF23
```

is unlikely to load the hex value ABCDEF23 into r3, for two reasons. First of all, the result of the addis instruction will be discarded by the second, since it sums ZERO + EF23, not the previous result. Better to use r3 as the second parameter. But it still won't work, because addi takes a SIGNED immediate operand, and EF23 sign-extends to FFFFEF23, not 0000EF23, which adds -1 to the previously loaded upper half. The correct sequence for loading ABCDEF23 into r3 is:

```
addis r3,0,0xABCE
addi r3,r3,0xEF23 or alternatively,

addi r3,0,0xEF23
addis r3,r3,0xABCE
```

or still better, because it's more understandable (ori takes an Unsigned operand):

```
addis r3,0,0xABCD
ori r3,r3,0xEF23
```

2. I'm not sure how Bill intends to use the -ze variants for add and subtract "as register-to-register move or negate and move mnemonics" but he's likely to be surprised when he tries to do so and finds the previous contents of the carry flag (XER.CA) randomizing his results somewhat. Better to stick to ORI for move, and NEG for negate and move.

3. Bill tells us that "Divide operations treat rA as a 64-bit dividend..." Perhaps somebody should tell Motorola, because their manual reports the much more reasonable proposition that the dividend is 32 bits. If it's 64, where do the other 32 bits come from?

4. It's really too bad we are stuck with the IBM syntax for the rotate operators. Or I should say YOU are stuck with it: very early on I realized I was, like Bill, burning a lot of time on this stuff, and altered my assembler and disassembler to reflect what is REALLY going on. All three of the rotates have very simple semantics: they rotate the source operand left n bits, then replace some bits in the destination with the rotated bits under the control of a mask. The remaining bits are either zeroed or left unchanged (the fundamental difference between the rlwimi and rlwinm). The problem is specifying the mask. See how much simpler these two instructions are to read:

```
rotm r29,r27,#3,=0007FFF8 ; rotate left 3, replace indicated bits
rotz r6,r15,#1,=00000080 ; rotate left 1, pick out a single bit
```

when compared to:

```
rlwimi r29,r27,3,13,28
rlwinm r6,r15,1,24,24
```

5. The latency figures Bill gives for branch instructions are likely to be misleading – perhaps this is why everybody makes the case for compilers being better. Branches are free if you give them enough setup time, basically three integer instructions after the one that altered the CR or CTR or LR register the branch depends on, but a sequence of branches with no data dependencies has a different kind problem. After a sequence of integer operations, the fourth consecutive branch not taken will introduce a bubble in the pipeline, for an effective 1 cycle delay. Consecutive branches taken cost two cycles each; they become free only if two or more integer operations separate each pair of branches taken. Then there are boundary conditions, but these three rules make for pretty efficient code.

6. I think Bill temporarily forgot that IBM numbers the 601 bits Big-Endian when he illustrated the mtrcr instruction. If CRM = 0x08, then it's cr4 (not cr3) that is replaced with bits 16-19 (not 12-15). He got the visual image correct, but he would be surprised when he went to use the bits by number. Another argument for the superiority of a visual mask over bit numbers. And yes, my assembler lets me use a visual mask syntax here as in the rotates. Perhaps somebody will come up with a macro preprocessor for the MPW assembler to parse the bit image syntax into something the assembler understands.

— Tom Pittman, *Itty Bitty Computers*

Bill Karsh responds... I am grateful to Tom Pittman for scrutinizing my article in such detail, and pleased that the readers and I will benefit from the corrections. I agree with Tom on most of his points, but let me respond to each.

- 0) High Level vs. Assembly programming – To everybody (not just

Tom who hates tired dogma) maybe I was not clear enough on my personal feelings about assembly. First of all, there is absolutely no question that just about anything coded in (good) PPC assembly can beat the pants off the best compiler yet available and probably ever likely to be available. I never could have intended otherwise. In fact, I code in assembly myself, but my particular work demands peak performance for a handful of core operations. It takes a great deal of effort to achieve this, and one can always improve the code by small changes here and there in a never ending process of refinement. If your particular job specification is to speed up existing and otherwise correct code, you can do much in C, but you can always do more in assembly by paying the price of being absolutely tied to machine-specific code. That's fine if you think it's worth the time that could be spent writing new, more portable and maintainable code. Yes, sometimes it is worth it. The optimization should be well targeted in any case.

What I wanted to argue about compilers was that the capability is there in the hardware to ease the compiler writer's job of optimizing. It ought to be possible for compilers to do better than they do today at PPC code and better at PPC code than they have ever been at 68K code. Since there is so much for you to do just to get your project on its feet, personally optimizing things should be a lower priority than making them correct and meeting specs. You will gain (some) optimization implicitly as the compilers improve, and there is some reason to be optimistic about this happening. Don't forget that as the machines get faster, the need for touch-ups keeps diminishing. There will always be a place for some killer assembly or some compiler hand-holding, but the genuine need should not arise as often as it used to. A blanket statement about assembly or optimization being evil would just be foolish.

1) Loading 0xABCDEF23 into a register – Tom is correct. My example is the result of hastily copying notes from place to place and incurring typos, for which I have no excuse. Each of his examples of loading a long literal constant is correct.

2) Clever uses of addze and subze as moves – Of course, the carry bit would have to be cleared for the moves to work as suggested by me. Tom is right again. If writing one's own assembly, his are preferred methods for effecting the moves reliably. Otherwise, the ze instructions should really only be employed for extended arithmetic.

3) The sizes of divw rD, rA, rB operands and results – I have no argument with Tom here either. The numerator (N), denominator (d) and quotient (Q) are all 32-bit quantities. When I said what I did about N being treated as 64-bits, I was merely likening the division to that familiar in the 68K divs.w instruction, where N is exactly twice the width of the d or Q registers. I intended that you might consider N in your mind as extended in this way as a formal convenience, not that the hardware operates this way.

4) Shift and rotate semantics – I think Tom is saying that he has created some simplifying macros for himself, which can only be lauded. However, I was concerned in the article with interpreting what most users are likely to see in their standard disassembler's output.

5) Branch timing – I agree only in spirit. There is much to say about branch timing. I reported a latency of one cycle for branch execution which is generally true – that's how long a branch takes to execute (in vacuum, so to speak). This gives little hint that a variety of things can happen depending on the context of the branch. I take issue with Tom's trying to characterize timing based on the language of branches taken or not taken. Those are the rules for 68K branch timing. On the PPC that is too simplistic. Branch timing is mainly governed by whether branches are correctly or incorrectly predicted. Incorrectly predicted branches hurt something awful, causing the IQ to be flushed, everything contingently executed to be flushed and new instructions to be fetched. This can cause a delay of more than one or two cycles. Further, the BPU handles one branch at a time, which is

why stacking them up is a no-no. The rules for employing branching to best advantage are complicated – too much so to be meaningfully summarized in the space of a letter.

6) mtrcr mask bits – Yep, I mistakenly reversed the bit numbering in the CRM mask parameter. The left-most bit of CRM corresponds to the left-most CR field (cr0) and similarly the right-most bit <-> right-most field (cr7). Whoops!

Let me elaborate on one thing that can be confusing and that occurs frequently in code. The extsb (sign extend byte) instruction extends to a width of 32-bits, unlike the ext.b 68K instruction which extends a byte to 16 bits. This behavior is in keeping with the idea that PPC arithmetic instructions act in general on all the whole of a register.

If anything else is annoying or just plain wrong in the presentation, let me hear about it.

– billKarsb@aol.com

OPENDOC, OLE, AND REAL DEVELOPERS

I have just received Microsoft's OLE SDK (free of charge) and been browsing it. There is a lot of marketing (evangelizing?) stuff in it, including some "deep" technical comparisons between OLE and OpenDoc. If you program the Macintosh, your future is OpenDoc – Apple says.

Well, Microsoft has different plans. If you program for OLE, which is available now, and it's free of charge, you can port your components to Windows **and** you can work with Excel or Word **now** – B. Gates says. After much reading and studying I came to a conclusion. I will support OpenDoc because frankly I don't care nor like Windows and I do vertical apps for Macs and UNIX, but if I was a mainstream developer I would go for OLE.

There are some technical differences. According to MS, OLE is a superior technology now and it will get better in the future. OpenDoc has several technical merits but, alas, it's not yet available. OpenDoc has a HUGE advantage too: It's **open**, and that means that source code is available and it's going to be ported from PDAs to Mainframes. Microsoft says that OLE is cross-platform (Win-Mac) now and it's true, and it says it will run under UNIX for free only if you licence (surprise) its Win32 API!!! And they call that Open. Please don't make me laugh.

I would like to see some input about ISD future plans on this technologies and some technical comparisons too.

So, take a pick, because we are going to start coding "parts" and putting them together like chips in a computer.

– Daniel Nofal Tech S.A Buenos Aires, ARGENTINA

DYLAN TAKES A LOAD OFF

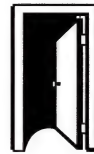
Thanks for the Sept. Dylan article. I was disappointed, though, that the article didn't use the example code to emphasize what makes Dylan different from C++ and other static languages. I'm not sure how many readers would wade through the code to discover the link between the interface definition of the OpenMovieFile function:

```
function "OpenMovieFile", output-argument: resRefNum;
and its invocation in the open-movie-file method:
```

```
let (err,ref-num) = OpenMovieFile(spec, file.data-permission);
nor notice some of the pleasures of Dylan they illustrate. err and
ref-num didn't have to be declared prior to their use – Dylan figured
it out from the context and created the properly-typed objects.
OpenMovieFile() is returning multiple values. The developer didn't
have to concern herself whether arguments should be passed by value
or by reference, nor worry about the intricacies of memory
management because Dylan has automatic garbage collection.
```

– Steve Palmen, tshirt2b@balcyon.com





By Chris Espinosa, Apple Computer, MacTech Magazine Regular Contributor

Think Like a Moviemaker, Part II

You may have a roster that really does look like movie credits...

Last month I proposed a style of software management that looked more like making a movie than writing an application. I suggested that, like making movies and TV shows, in the future programmers, interface designers, testers, and architects will move from company to company and project to project like actors, makeup artists, and set designers.

For those of us who want to believe that software development is an art, this might be upsetting: how can you create a well-crafted, tight, pleasant-to-use application with a bunch of people whose only work experience together will be the twelve to eighteen months they spend on this release? Where's the art in a bunch of contractors dog-piling on one idea?

Simple – the art of it is the same as the art of Schindler's List, or Forrest Gump, or E.T. It's in a shared, industrywide ethic of relationships over rivalries; in a guild system where all crafts are valued, but not ranked against each other; and it's in a management system where the process is understood by everybody so all can focus on the quality of their work.

Managing a product produced like this will be different from managing organizational software development today. First, you'll start with the concept before you get the funding or the people. That will take a producer, who will stay with the project its entire lifespan, and some architects – the software equivalent of scriptwriters – who will write the story that the software tells.

Who's the audience for the program? What are their deepest needs? How will

your program touch them in a way no software has done before? These are usually considered "marketing" questions, but they're really the key creative questions of the project. No market research can tell you what people want if they have never seen it; that's why you need people who know the possibilities of the medium, people who appreciate the great things others have done, and people who understand structurally why other products are good, useful, and successful in order to define what you want to do.

Then the architects need to sketch out the plot of the application. What are the major functions and the subfunctions that run among them? What does a day's work using your program look like? It might help to literally write a script here, a sort of dialog between the user and the computer. Try to make the interaction as smooth as possible. Listen for the "clunkers" in the dialog where a real live person would never say or do that. Strive for a natural feel to the rhythm of work.

Then go onto rewrites. Have a bunch of knowledgeable people read the script and critique it honestly. Get real users to read it. Maybe mock up the interface in Director or HyperCard to get the key points across.

When your screenplay is well along, it's time to hire a director and the principal programmers. You'll choose people, of course, who know about the genre of program you're writing (hiring a graphics person to do a mainframe front-end would be like hiring Ron Howard to do a shoot-'em-up). Make these people your core team and let them run the rest of the casting of programmers. The producer should start to pick the behind-the-camera people (testers, configuration management, interface design, documentation, marketing, support).

It's now time to start real project planning. This is where you turn the screenplay into a shooting script: take each module and go into detail for what you need from each department. Real planning expertise is crucial here. If you can plan the whole project module by module and know what resources are needed where, you'll have a much better chance of keeping the quality high and bringing the program in on schedule. If you make it up as you go along, you'll hit resource conflicts, cascading delays, and a kind of paralysis where nobody's getting any work done because they're trying to cope with an out-of-control project.

And watch that schedule. Of course things will not come in on time or budget, some modules will slip, and things won't happen as expected. After writing a module you may find it just doesn't fit into the plot that well. Reshoot it. Don't just rewrite it: start with the screenplay, write the scenarios again, and see if you can find a more satisfying idea. Meanwhile have the principal programmers shoot a different scene, instead of having them idle, or fiddling around doing the scriptwriter's work.

Finally, take time in post-production. Maybe here is the time to let

**NOW YOU CAN ADD MOVIE-LIKE
GRAPHIC EFFECTS
TO YOUR MAC APPLICATIONS**

QDFx™

•WIPEs • DISSOLVES • TRANSITIONS & MORE

Already part of a new multimedia authoring tool, Ariel Publishing's latest release of "QDFx", includes a library of 20 graphic effects you can apply to your Mac applications and without the overhead of including a quick time movie.

It's fast, it's small, it's royalty free and only...

\$69.95

System 7 compatible, 32-bit clean, C source code available (sold separately). Native Power PC version available by June 15, 1994

ARIEL PUBLISHING INC.

To order by mail: P.O. Box 398
Pateros, WA 98846-0398

Phone: (509) 923-2249

E-mail: America Online - (ARIELPUB1),
GEnie - (InsideBasic),
CIS - (71441,2262)

Visa/MC and purchase orders cheerfully accepted.

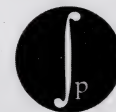
the principal programmers go do another project, while a crew of seasoned editors and debuggers, under the director's guidance, tune and debug the program in conjunction with the beta testers and focus groups. Just as Meryl Streep's talents would be wasted doing foleys of footsteps, your top-dollar programmers can find more productive and creative things to do than fix memory leaks or spelling errors in dialog boxes.

When you release your program, of course the director and lead programmers get the most credit, because even the best story can be made bad by weaknesses here. Next come the writers and producers, who came up with the idea and made it hang together. Next comes the supporting cast of programmers, then the leads in other areas, and finally all the people who participated. Credit for everybody is crucial. These people get their next job by the quality of the work they did on this one, and by their contact with a great project and great people. It's part of their resume, not just twelve forty-hour weeks.

Does this sound so different from the way you do things now? It mixes up the order a little; puts some non-programmers in terribly influential positions; assumes that one programmer can debug another's code; and doesn't even try to keep the team together for the next version. With some of today's tools and practices these could be disastrous. But by focusing on disciplined coding and documentation practice, modular component architecture, and better tools, you may be able to start putting some of these techniques into practice soon.



MacRegistry™
Developer Job Opportunities



If you are a Macintosh developer, you should register with us! We have a database that enables us to let you know about job opportunities. When we are asked to do a search by a client company the database is the first place we go. There is no charge for registering. The database service is free. Geographic Coverage is nationwide.

Marketability Assessment - To get a specific feel for your marketability send a resumé via Email or call. You may also request a Resume Workbook & Career Planner.

Discreet - We are very careful to protect the confidentiality of a currently employed developer.

Scientific Placement is managed by graduate engineers, we enjoy a reputation for competent & professional job placement services and we are Mac fanatics.

1-800-231-5920 | AppleLink: D1580 | 713-496-0373 fax

Scientific Placement, Inc.

Dept. MT MacRegistry
P.O. Box 19949
Houston, Texas 77224
(713) 496-6100
das@scientific.com

Dept. MT MacRegistry
P.O. Box 71
San Ramon, CA 94583
510-733-6168
bge@scientific.com

Dept. MT MacRegistry
P.O. Box 4270
Johnson City, TN 37602
(615) 854-9444
rjg@scientific.com

MACXPerts

MacXperts is a software development company which has immediate openings for people who are experienced MacApp programmers and for C++ programmers who want to learn MacApp or NTK. If you have what it takes, and the desire to achieve, call Kendall Tyler at MacXperts.

Phone 804-353-7122 **Fax** 804-358-3847

**Hiring
MacApp
Programmers**

**MacTech Magazine
is your
recruitment vehicle**

When you need to fill important positions at your company, MacTech Magazine is the consistent choice of companies across the country for hiring the best qualified Macintosh programmers and developers. Let MacTech Magazine deliver your recruitment message to an audience of over 27,000 qualified computer professionals.

**Call Ruth Subrin at
310/575-4343**

If you have a CD-ROM drive, then you've gotta have every article published in the first 8 years of MacTech Magazine, including all of 1992. Over 900 articles. All 95 issues. All the source code. THINK Reference 2.0 On Location 2.0 Working applications. Full documentation. Demos for developers. Free shareware code.

Plus, Plus, Plus.

FREE Upgrade to next version of CD-ROM!!*

*Any registered user who purchases *All of MacTech Magazine CD-ROM Volume 1-8* after November 1, 1993 will receive a free upgrade to the next version of the *All of MacTech Magazine CD-ROM* when available.

MacTech Formerly MacTutor **MAGAZINE™**
FOR MACINTOSH PROGRAMMERS & DEVELOPERS

Voice: 310/575-4343 • Fax: 310/575-0925
AppleLink: MT.CUSTSVCS • CompuServe: 71333,1063
Internet: custservice@xplain.com
America Online & GENie: MACTECHMAG

More than 500 megs, selected by Macintosh developers for Macintosh developers, all on the new All of MacTech Magazine CD-ROM, Volume 1-8.

In The MacTech Magazine Folder

- Every article published in the first 8 years
- Complete source code & working examples: Assembly, Pascal, C, FORTRAN, BASIC, Lisp, MacScheme, Forth, Neon, MOPS, Yerk, Prograph, Hypercard, Ada, 4th Dimension, MacApp, THINK Class Library
- Complete text of articles including dialogs, custom windows and menus, sounds, viruses, object oriented programming, animation, simulations, XCMDs and XFCNs, parsing, User Interface Design, List Manager, Class Libraries, debugging, product reviews, inits, control panel devices.
- Special ViewR document reader to view, search and print articles.
- And much more!

THINK™ Reference

FREE!

Symantec's THINK Reference 2.0. Complete on line guide to Inside Macintosh, Vol. I-VI, with cross referenced index, detailed information of each function, procedure and detail needed when programming the Macintosh.

SYMANTEC.™

In The Apple Folder

- "Up Your FCBs"
- BitMapToRegion
- CD-ROM 3.2
- Discipline
- Exec
- MacsBug 6.2.2
- Basic Connectivity Set w/Communications Toolbox
- System 7.0.1 with Tune-up
- Logic Manager 1.0d2
- ResEdit 2.1.1+
- QuickTime 1.5
- And much more!

FREE!

ON LOCATION™

A full working, registered version of On Location 2.0, from On Technology. Use it to index all your source code disks. Find references in seconds on multiple disks whether or not the disk is mounted.

Demos relevant to developers

- Demos for all kinds of third party tools and utilities

Public/Shareware Code

- 150+ meg, most are fully documented, including many updates
- Best of BMUG's Programming Tools, System 7 Utilities, etc.
- Hundreds of code ideas, explanations, help files
- Tutorials
- Source code for more than 70 working applications
- And much, much more!

More! More! More!

We could go on listing everything that is included on this new exciting CD-ROM. But you should see it for yourself. Order yours today, for \$299 (upgrades for \$150) plus shipping & handling. Call, fax, or e-mail for more information.



MACTECH MAGAZINE PRODUCTS & ORDER INFORMATION

E-mail, Fax, write, or call us. You may use your VISA, MasterCard or American Express; or you may send check or money order (in US funds only): MacTech Magazine, P.O. Box 250055, Los Angeles, CA 90025-9555. Voice: 310/575-4343 • Fax: 310/575-0925

If you are an e-mail user, you can place orders or contact customer service at:

- **AppleLink:** MT.CUSTSVC
- **CompuServe:** 71333,1063
- **Internet:** custservice@xplain.com
- **America Online:** MT CUSTSVC
- **GEnie:** MACTECHMAG

SUBSCRIPTIONS

US Magazine: \$47 for 12 issues
Canada: \$59 for 12 issues
International: \$97 for 12 issues
Domestic source code disk: \$77 for 12 issues
Int'l source code disk: \$97 for 12 issues

CD-ROM

All of MacTech Volumes I-VIII CD-ROM: Includes over 900 articles from all 95 issues (1984-1992) of MacTech Magazine (formerly MacTutor). All article text and source code. Articles and code are indexed by On Location. The CD includes Symantec's THINK™ Reference 2.0, On Technology's On Location™ 2.0, working applications with full documentation, product demos for developers and more. See advertisement, this issue: \$299. Upgrades \$150, e-mail, call or write for info.

BOOKS

The Best of MacTutor, Volume 1: **Sold Out**
The Complete MacTutor, Volume 2: **Sold Out**
The Essential MacTutor, Volume 3: \$19.95
The Definitive MacTutor, Volume 4: \$24.95
The Best of MacTutor, Volume 5: \$34.95
Best of MacTutor Collection, Volumes 3-5: \$69
Best of MacTutor, Volumes 6, 7, 8 & 9: Not available

DISKS

Source Code Disks: \$8 each
Topical Index (1984-1991) on disk: \$5

MAGAZINE BACK ISSUES

Volumes 3, 4, 5, 6, 7, 8, 9 and 10: \$5 each (subject to availability)

SHIPPING, HANDLING & TAXES

California:

Source disk or single issue: \$3
Single book or multiple back issues: \$5
Two books: \$8 • All other orders: \$12

California residents include 8.25% sales tax on all software, disks and books.

Continental US:

Source disk or single issue: \$3
Single book or multiple back issues: \$7
Two books: \$15 • All other orders: \$17

Canada, Mexico and Overseas: Please contact us for shipping information. Allow up to 2 weeks for standard domestic orders, more time for international orders.

PLEASE NOTE

Source code disks and journals from MacTech Magazine are licensed to the purchaser for private use only and are not to be copied for commercial gain. However, the code contained therein may be included, if properly acknowledged, in commercial products at no additional charge. All prices are subject to change without notice.

10% OFF
ALL BOOKS!

3RD PARTY PRODUCTS

MACTECH EXCLUSIVES

MacTech Magazine is your exclusive source for these specific products:

NEW! Ad Lib 2.0 The premier MacApp 3.0 compatible ViewEdit replacement. A powerful user-interface editing tool to build views for MacApp 3.0 and 3.1. Ad Lib allows subclassing of all of MacApp's view classes including adorners, behaviors, and drawing environments. String and text style resources are managed automatically. Alternate display methods, such as a view hierarchy window, allow easy examination of complex view structures. Ad Lib includes source code for MacApp extensions that are supported by the editor - buttons can be activated by keystrokes, behaviors can be attached to the application object, and general purpose behaviors can be configured to perform a number of useful functions. Run mode allows the user to try out the views as they will work in an application. Templates can be created to add additional data fields to view classes. Editing palettes provide fast and easy editing of common objects and attributes. Works with ACI's Object Master (version 2.0 and later) to navigate a project's user interface source code. \$195

NEW! FrameWorks Magazine: \$8/issue, subject to availability.

NEW! FrameWorks Source Code Disk: \$10/issue, subject to availability.

NEW! Five Years of Objects CD-ROM: FrameWorks archives and source code from April 1991 to January 1993, plus selected object-oriented publicly available software and demos. \$95

MADACON '93 CD-ROM: The highlights of MADACON '93, including Mike Potel on Pink, Bedrock, MacApp, OODLs, and more. Slides, articles, demos, audio, and QuickTime. \$95

NEW! MAScript 1.2 adds support for AppleScript to your MacApp 3.0.1 and 3.1 based applications. Make your application scriptable and recordable by building on a tried and tested framework for object model support. MAScript dispatches Apple events to the appropriate objects, creates object specifiers, and makes framework objects like windows and documents scriptable and recordable. Sample

application shows you how to begin adding support for scripting and recording. MAScript includes complete source code. Install MAScript by modifying one MacApp source file, then adding another to your project. Future versions of MacApp will incorporate MAScript, so MAScript support you add now will work in the future. \$199

NEW! The Mjølner BETA System is a software development environment supporting object-oriented programming in the BETA programming language. BETA is uniquely expressive and orthogonal. BETA unifies just about every abstraction mechanism - including class, procedure, function, coroutine, process and exception - into the ultimate abstraction mechanism: the pattern. BETA includes: general block structure, strong typing, whole/part objects. The compiler: binary code generation, automatic garbage collection, separate compilation, interface to C, Pascal, and assembler.

The system: persistent objects, basic libraries with containers classes, platform-independent GUI application frameworks on Unix, Mac and Windows NT, metaprogramming system. The tools available on Unix: the hyper structure editor supporting syntax directed editing, browsing, etc., and the source code debugger are currently being ported to the Macintosh system. The Mjølner BETA System for Macintosh requires MPW (basic set) 3.2 or later.

Package containing compiler, basic libraries, persistent store, GUI framework, and comprehensive documentation. (Other packages are also available) \$295

NEW! Savvy OSA support includes attachability, recordability, scriptability, coercion, in addition to script execution, idling and i/o. Apple event support includes complex object specifiers, synchronous/asynchronous Apple event handling, and Apple event transactions for clients and servers. The Core Suite of Apple event objects is supporting including the application, documents, windows, and files. Documentation includes technology overview, cookbook, and sample code. \$250

NEW! More Savvy includes all Savvy features plus Apple event support for all sub-classes of TEventHandler with extensive view support. Apple event support for text includes text attributes and sub-range specification. Recordability supports

MAIL ORDER STORE

additional actions, and coercion includes additional types. Additional client and server Apple events. \$450

NEW! Super Savvy includes all More Savvy features plus compile, edit, and record scripts using built in script editor. View template editors, like Ad Lib, can attach scripts to view objects and modified scripts are saved with the document. Script action behavior allow quick access for executing and editing scripts attached to views. Text to object specifier coercion plus more. \$700

NEW! Savvy QuickTime Requires Savvy, More Savvy, or Super Savvy. Includes QuickTime, Apple event and view template support. Movies come out of the box ready to play, edit, and react to Apple events. They can be included in any view structure, including templates, and are displayed in the scrap view. Movie controls include volume, play rate, looping mode, display style, and other characteristics. \$250

NEW! Savvy DataBase Requires Savvy, More Savvy, or Super Savvy. Available Winter 1994. \$250

MacTech Magazine is your exclusive source for available back issues of SFA's magazine, source code disks and assorted CD's. Call for more info and pricing.

BOOKS

Defying Gravity: The Making of Newton Doug Menuez and Markos Kounalakis. An in depth, dramatic account of the story of Newton's creation. It is a techno-logical adventure story; a fascinating case study of the process by which an idea is born and then translated into a product on which careers and fortunes can be made or lost. It is a new kind of business book, one that captures through powerful photo-journalism and a fast-paced text, the human drama and risk involved in the invention of a new technology for a new marketplace. 196 pgs., \$29.95 \$26.95

The Elements of E-Mail Style by Brent Heslop and David Angell. Learn the rules of the road in the e-mail age. Concise, easy-to-use format explaining essential e-mail guidelines and rules. It covers style, tone, typography, formatting, politics and etiquette. It also outlines basic rules of composition within the special context of writing e-mail and includes samples and templates for writing specific types of e-mail correspondence. 208 pages. \$14.95 \$13.45

NEW! E-Mail Essentials by Ed Tittel & Margaret Robbins is a hands-on guide to the basics of e-mail, the ubiquitous networks communication system. The book is suitable for both the casual e-mailer and the networking professional, as it covers everything from the installation of e-mail to the maintenance and management of e-mail hubs and message servers. The books explains the fundamental concepts and technologies of electronic mail, featuring chapters on Lotus applications and CompuServe, as well as information on upgrading, automation, message-based applications, and user training. E-mail is a step-by-step, jargon-free guide that will enable the e-mail user to get the most out of the communication potentials of networking. 250 pp. \$24.95 \$22.45

NEW! Graphics Gems IV edited by Paul Heckbert Volume IV is the newest collection of carefully crafted, innovative gems. All of the gems are immediately accessible and useful in formulating clean, fast, and elegant programs. The C programming

language is used for most of the program listings, although several of the gems have C++ implementations. An IBM or Macintosh disk containing all of the code from all four volumes is included. Includes one 3.5" high-density disk. \$49.95 \$44.95

How To Write Macintosh Software by Scott Knaster is a great source for understanding Macintosh programming techniques. Drawing from his years of experience working with programmers, Scott explains the mysteries and myths of Macintosh programming with wit and humor. The third edition, fully revised and updated, covers System 7 and 32-bit developments, and explores such topics as how and where things are stored in memory; what things in memory can be moved around and when they may be moved; how to debug your applications with MacsBug; how to examine your program's code to learn precisely what's going on when it runs. 448 pgs., \$29.95 \$26.05

The Instant Internet Guide by Brent Heslop and David Angell. An Internet jump-start - how to access, use and navigate global networks. The Instant Internet Guide equips readers with the tools needed to travel the electronic world. The book highlights the most important sources of Internet news and information and explains how to access information on remote systems. It outlines how to use essential Internet utilities and programs and includes a primer on UNIX for the Internet. 224 pages \$14.95 \$13.45

Learn C on the Macintosh by Dave Mark. This self-teaching book/disk package gives you everything you need to begin programming on the Macintosh. Learn to write, edit, compile, and run your first C programs through a series of over 25 projects that build on one another. The book comes with THIN C - a customized version of Symantec's THINK C, the leading programming environment for Macintosh. 464 pages, Book/disk: \$34.95 \$31.45

Learn C++ on the Macintosh by Dave Mark. After a brief refresher course in C, Learn C++ introduces the basic syntax of C++ and object programming. Then you'll learn how to write, edit, and compile your first C++ programs through a series of programming projects that build on one another as new concepts are introduced. Key C++ concepts such as derived classes, operator overloading, and iostream functions are all covered in Dave's easy-to-follow approach. Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages, \$36.95 \$33.26

Macintosh C Programming Primer Volume I, Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed. This new edition of this Macintosh programming bestseller is updated to include recent changes in Macintosh technology, including System 7, new versions of THINK C and ResEdit, and new Macintosh machines. Readers will learn how to use the resources, Macintosh Toolbox and interface to create stand-alone applications. 672 pages, \$26.95 \$24.25

Macintosh C Programming Primer Volume II, Mastering the Toolbox Using THINK C by Dave Mark. Volume II picks up where Volume I leaves off, covering more advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager. 528 pgs. \$26.95 \$24.25

Macintosh Pascal Programming Primer Volume I, Inside the Toolbox Using THINK Pascal by Dave Mark and Cartwright Reed. This tutorial shows programmers new to the Macintosh how to

use the Toolbox, resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages \$26.95 \$24.25

Macintosh Programming Techniques by Dan Sydnor (Series Editor: Tony Meadow). This tutorial and handbook provides a thorough foundation in the special techniques of Macintosh program-ming for experienced Macintosh programmers as well as those making the transition from DOS, Windows, VAX or UNIX. Emphasizes programming techniques over syntax for better code, regardless of language. Guides the reader through Macintosh memory management, QuickDraw, events and more, using sample program in C++. Disk includes an interactive tutorial, plus reusable C++ code. \$34.95 \$31.95

NEW! Multimedia Authoring Building and Developing Documents by Scott Fisher addresses the concerns that face anyone trying to create multimedia documents. It offers specific advice on when to use different kinds of information architecture, discusses the human-factors concepts that determine how readers use and retain information, and then applies these findings to multimedia documents, covering the high-level issues concerning planners and authors of multimedia documents as well as those involved in evaluating or purchasing multimedia platforms. Includes one 3.5" high-density disk. \$34.95 \$31.45

NEW! Programming for the Newton Software Development with NewtonScript by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith. Programming for the Newton: Software Development with NewtonScript is an indispensable tool for Newton programmers. Readers will learn how to develop software for the Newton on the Macintosh from people that developed the course on programming the Newton for Apple Computer. The enclosed 3.5" disk contains a sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK), Apple Computers complete development environment for the Newtons. A Publication of AP Professional May 1994, Paperback, 393 pp. \$29.95 \$26.95

Programming in Symantec C++ for the Macintosh by Judy May and John Whittle. This book will introduce you to object-oriented programming, the C++ language, and of course Symantec C++ for the Macintosh. You don't have to be a programmer, or even know anything about programming to benefit from this book. Programming in Symantec C++ for the Macintosh covers everything from the basics to advanced features of Symantec C++. If you are a Think C or Zortech C++ programmer who wants to learn more about object-oriented programming or what's different about Symantec C++, there are whole chapters specifically for you. Includes helpful examples of C++ code that illustrate object-oriented programs. \$29.95 \$26.95

Programming for System 7 by Gary Little and Tim Swihart, is a hands-on guide to creating applications for System 7. It describes the new features and functions of the operating system in detail. Topics covered include file operations, cooperative multitasking, Balloon Help, Apple events, and the File Manager. Numerous working C code examples show programmers how to take advantage of each of these features and use them in developing their applications. 384 pages \$26.95 \$24.25

ResEdit™ Complete, Second Edition by Peter Alley and Carolyn Strange. With ResEdit, Macintosh programmers can customize every aspect of their interface

Want more product info? Call us at 310/575-4343.

form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package. ~~\$34.95~~ **\$31.45**

Sad Macs, Bombs, Disasters and What to Do About Them by Ted Landau comes to the rescue with your Macintosh problems. From fractious fonts to the ominous Sad Macintosh icon, this emergency handbook covers the whole range of Macintosh problems: symptoms, causes, and what you can do to solve them. 640 Pages ~~\$24.95~~ **\$22.45**

Software By Design: Creating User Friendly Software by Penny Bauersfeld (Series Editor: Tony Meadow). This excellent reference provides readers with a thorough how-to for designing software that is easy to learn, comfortable to operate and that inspires user confidence. Written from the perspective of Macintosh, but compatible with all platforms. Stresses user input from initial design, through prototyping, testing and revision. Provides tools for analyzing user needs and test responses. Includes exercises for sharpening user-oriented design skills. ~~\$29.95~~ **\$26.95**

NEW! Taligent's Guide to Designing Programs Well-Mannered Object-Oriented Design in C++ is the Taligent approach to object-oriented design. The Taligent Operating Environment is the first commercial software system based entirely on object-oriented technology. Taligent's Guide to Designing Programs is a developer's-eye view of this system. It introduces new concepts of programming and empowers developers to create software more productively. Out of their direct experience in developing the system, the authors focus on global issues of object-oriented design and writing C++ programs, and the specific issues of programming in the Taligent Operating Environment. Taligent's Guide to Designing Programs assumes the reader is an experienced C++ programmer, and proceeds from there to fully explore "the Taligent way" of programming. ~~\$19.50~~ **\$17.55**

Writing Localizable Software for the Macintosh by Daniel R. Carter. 469 pages. ~~\$26.95~~ **\$24.25**

THE APPLE LIBRARY

HyperCard Stack Design Guidelines by Apple Computer, Inc. is an essential book for everyone who creates Apple HyperCard stacks, from beginners to commercial developers. It covers the basic principles of design that, when incorporated, make HyperCard stacks effective and usable. Topics include guidelines, navigation, graphic design and screen illustration, text in stacks, music and sound, a sample stack development scenario, collaborative development, and the Stack Design Checklist. 240 pages, ~~\$24.95~~ **\$19.95**

Inside AppleTalk by Gursharan S. Sidhu, Richard F. Andrews and Alan B. Oppenheimer. Apple Computer, Inc. 650 pages, ~~\$34.95~~ **\$31.45**

Inside Macintosh: AOC Service Access Interfaces by Apple Computer, Inc. shows how your application can take advantage of the system software features provided by PowerTalk system software and the PowerShare collaboration servers. Nearly every Macintosh application program can benefit from the addition of some of these features. This book shows how you can add electronic mail capabilities to your application, write a messaging application or agent, store information in and

retrieve information from PowerShare and other AOC catalogs, add catalog-browsing and find-in-catalog capabilities to your application, write templates that extend the Finder's ability to display information in PowerShare and other AOC catalogs, add digital signatures to files or to any portion of a document, and establish an authenticated messaging connection. ~~\$40.45~~ **\$36.40**

Inside Macintosh: AOC Service Access Modules by Apple Computer, Inc. describes how to write a software module that gives users and PowerTalk-enabled applications access to a new or existing mail and messaging service or catalog service. This book shows how to write a catalog service access module (CSAM), a messaging service access module (MSAM), and AOC templates that allow a user to set up a CSAM or MSAM and add addresses to mail and messages. ~~\$26.95~~ **\$24.25**

NEW! Inside Macintosh: CD-ROM by Apple Computer, Inc. is the essential guide to using AppleScript to customize and automate the Macintosh. Inside Macintosh® is the essential reference for programmers, designers, and engineers for creating applications for the Macintosh family of computers. Inside Macintosh CD-ROM collects more than 25 volumes in electronic form, including: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components. Now programmers will be able to access over 16,000 pages of the information they need directly from their computers. Hypertext linking and extensive cross referencing across volumes allows programmers to search and explore this library in ways that are unique to the electronic medium. Every Macintosh programmer will regard Inside Macintosh CD-ROM as their most important resource. \$99.95

Inside Macintosh: Devices by Apple Computer, Inc. describes how to write software that interacts with built-in and peripheral hardware devices. With this book, you'll learn how to write and install your own device drivers, desk accessories, and Chooser extensions; communicate with device drivers using the Device Manager; access expansion cards using the Slot Manager; control SCSI devices using SCSI Manager 4.3 or the original SCSI Manager; communicate directly with Apple Desktop Bus devices; interact with the Power Manager in battery-powered Macintosh computers; and communicate with serial devices using the Serial Driver. ~~\$29.95~~ **\$26.95**

Inside Macintosh: Files by Apple Computer, Inc. describes the parts of the operating system that allow you to manage files. It shows how your application can handle the commands typically found in a File menu. It also provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs, ~~\$29.95~~ **\$26.95**

Inside Macintosh: Interapplication Communication by Apple Computer, Inc. shows how applications can work together. How your application can share data, request information or services, allow the user to automate tasks, communicate with remote databases. ~~\$34.95~~ **\$31.45**

Inside Macintosh: Imaging by Apple Computer, Inc. covers QuickDraw and Color QuickDraw. The book includes general discussions of drawing and working with color. It describes the structures that hold images and image information, and the routines that manipulate them. It also covers the Palette, Color, and Printing Managers,

MAIL ORDER STORE

and the Color Picker, Color Matching, and Picture Utilities. ~~\$26.95~~ **\$24.25**

Inside Macintosh: Macintosh Toolbox Essentials by Apple Computer, Inc. covers the heart of the Macintosh. The toolbox enables programmers to create applications consistent with the Macintosh "look and feel". This book describes Toolbox routines and shows how to implement essential user interface elements, such as menus, windows, scroll bars, icons and dialog boxes. 880 pages ~~\$24.95~~ **\$31.45**

Inside Macintosh: More Macintosh Toolbox by Apple Computer, Inc. covers other Macintosh features such as how to support copy and paste, provide Balloon Help, play and record sound and create control panels are covered in this volume. The managers discussed include Help, List, Resource, Scrap and Sound. ~~\$34.65~~ **\$31.45**

Inside Macintosh: Memory by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to manage memory. It provides detailed strategies for allocating and releasing memory, avoiding low-memory situations, reference to the Memory Manager, the Virtual Memory Manager, and memory-related utilities. 296 pages, ~~\$24.95~~ **\$22.45**

Inside Macintosh: Networking by Apple Computer, Inc. describes how to write software that uses AppleTalk networking protocols. It describes the components and organization of AppleTalk and how to select an AppleTalk protocol. It provides the complete application interfaces to all AppleTalk protocols, including ATP (AppleTalk Transaction Protocol), DDP (Datagram Delivery Protocol), and ADSP (AppleTalk Data Stream Protocol), among others. ~~\$29.95~~ **\$26.95**

Inside Macintosh: Operating System Utilities by Apple Computer, Inc. describes parts of the Macintosh Operating System that allow you to manage various low-level aspects of the operating system. Everyone who programs the Macintosh should read this book! It will show you in detail how to get information about the operating system, manage operating system queues, handle dates and times, control the settings of the parameter RAM, manipulate the trap dispatch table, and receive and respond to low-level system errors. ~~\$26.95~~ **\$23.45**

Inside Macintosh: Overview by Apple Computer, Inc. is the first book that people who are unfamiliar with Macintosh programming should read. It gives an overview of Macintosh programming fundamentals and a road map to the New Inside Macintosh library. Inside Macintosh: Overview also covers various programming tools and languages, compatibility guidelines and an overview of considerations for worldwide development. 176 pages, ~~\$22.95~~ **\$20.65**

Inside Macintosh: PowerPC Numerics by Apple Computer, Inc. describes the floating-point numerics environment provided with the first release of PowerPC processor-based Macintosh computers. The numerics environment conforms to the IEEE standard 754 for binary floating-point arithmetic. This book provides a description of that standard and shows how RISC Numerics compiles with it. This book also shows programmers how to create floating-point values and how to perform operations on floating-point values in high-level languages such as C and in PowerPC assembly language. ~~\$28.95~~ **\$26.00**

Inside Macintosh: PowerPC System Software by Apple Computer, Inc. describes the new process execution environment and system software

Want more product info? E-mail us at productinfo@xplain.com

MAIL ORDER STORE

services provided with the first version of the system software for Macintosh on PowerPC computers. It contains information you need to know to write applications and other software that can run on the PowerPC. PowerPC System Software shows in detail how to make your software compatible with the new run-time environment provided on PowerPC-based Macintosh computers. It also provides a complete technical reference for the Mixed Mode Manager, the Code Fragment Manager, and the Exception Manager. ~~\$24.95~~ **\$22.45**

Inside Macintosh: Processes by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to control the execution of processes and interrupt tasks. It shows in detail how you can use the Process Manager to get information about processes loaded in memory. It is also a reference for the Vertical Retrace, Time, Notification, Deferred Task, and Shutdown Managers. 208 pages, ~~\$22.95~~ **\$20.65**

Inside Macintosh: QuickTime by Apple Computer, Inc. is for anyone who wants to create applications that use QuickTime, the system software that allows the integration of video, animation, and sounds into applications. This book describes all of the QuickTime Toolbox utilities. In addition, it provides the information you need to compress and decompress images and image sequences. ~~\$29.95~~ **\$26.95**

Inside Macintosh: QuickTime Components by Apple Computer, Inc. covers how to use and develop QuickTime components such as image compressors, movie controllers, sequence grabbers, and video digitizers. ~~\$34.95~~ **\$31.45**

Inside Macintosh: Sound by Apple Computer, Inc. describes the parts of the Macintosh system software that allow you to manage sounds. It contains information that you need to know to write applications and other software that can record and play back sounds, compress and expand audio data, convert text to speech, and perform other similar operations. ~~\$26.95~~ **\$24.25**

Inside Macintosh: Text by Apple Computer, Inc. describes how to perform text handling, from simple character display to multi-language processing. The Font, Script, Text Services, and Dictionary Managers are all covered, in addition to QuickDraw Text, TextEdit, and International and Keyboard Resources. ~~\$39.95~~ **\$35.95**

Inside Macintosh: QuickDraw™ GX Library by Apple Computer, Inc. is the powerful new graphics architecture for the Macintosh. Far more than just a revision of QuickDraw, QuickDraw GX is a unified approach to graphics and typography that gives programmers unprecedented flexibility and power in drawing and printing all kinds of shapes, images, and text. This long-awaited extension to Macintosh system software is documented in a library of books that are themselves an extension to the new Inside Macintosh series. The QuickDraw GX Library is clear, concise, and organized by topic. The books contain detailed explanations and abundant programming examples. With extensive cross-references, illustrations, and C-language sample code, the QuickDraw GX Library gives programmers fast and complete reference information for creating powerful graphics and publishing applications with sophisticated printing capabilities. The first two volumes in the QuickDraw GX Library are:

Inside Macintosh: QuickDraw GX Objects by Apple Computer, Inc. introduces QuickDraw GX and its object structure, and shows programmers how to manipulate

objects in all types of programs. ~~\$26.95~~ **\$24.25**

Inside Macintosh: QuickDraw GX Graphics by Apple Computer, Inc. shows readers how to create and manipulate the fundamental geometric shapes of QuickDraw GX to generate a vast range of graphic entities. It also demonstrates how to work with bitmaps and pictures, and specialized QuickDraw GX graphic shapes. ~~\$26.95~~ **\$24.25**

NEW! Inside Macintosh: X-Ref. by Apple Computer, Inc. is an index for Inside Mac. ~~\$12.95~~ **\$11.65**

LANGUAGES



New Pricing!

CodeWarrior™ CD by Metrowerks comes in two versions — Bronze and Gold. These CDs contain the CodeWarrior development environment including C++, C and Pascal compilers; high-speed linkers; native-mode interactive debuggers; and a powerful new application framework called PowerPlant for rapid Macintosh development in C++. Bronze generates 680x0 code. Gold generates both 680x0 and PowerPC code. All versions are a 3 CD subscription over a 1-year period. Bronze: \$99, Gold: \$399. **Bronze comes with a 6-month MacTech subscription. Gold comes with a 1-year subscription. Both at no additional charge!**



NEW!

Geekware by Metrowerks is here! In high school, they called you a computer geek. Now, they work at burger joints and you don't. Wear it to your favorite burger joint. ~~\$24.95~~



FORTRAN by Language Systems is a full-featured ANSI standard FORTRAN 77 compiler that runs in the Macintosh Programmers Workshop (MPW). All major VAX extensions are supported as well as all major features of Cray and Data General FORTRAN. FORTRAN creates System 7 savvy applications quickly and easily. Compiler options specify code generation and optimization for all Macintoshes, including special optimizations for 68040 machines. Error messages are written in plain English and are automatically linked to the source file. The runtime user interface of compiled FORTRAN programs is fully customizable by programmers with any level of Macintosh experience. \$595. w/o MPW: \$495. Corporate 5 pack \$1575

FORTAN 77 SDK for Power Macintosh by Absoft includes a globally optimizing native compiler and linker, native Fx™ multi-language debugger, and Apple's MPW development environment. The compiler is a full

ANSI/ISO FORTRAN 77 implementation and includes all MIL-STD 1753 extensions, Cray/Sun-style POINTER, and several Fortran 90 enhancements. MRWE, Absoft's application framework libraries, is included as is the MIG graphics library for quick creation of plots and graphs. The native Macintosh PPC toolbox is fully supported. Absoft's Fx debugger can debug intermixed FORTRAN 77, C, C++, PPC assembler. The compiler, linker, and debugger all run as native PPC tools and produce native Macintosh PPC executables. \$699

MacFortran® II V3.3 is a VAX/VMS compatible, full ANSI/ISO FORTRAN 77 compiler including all MIL-STD 1753 extensions. Acknowledged to be the fastest FORTRAN available for Macintosh, MacFortran II is bundled with the latest version of Macintosh Programmer's Workshop (MPW), and includes SourceBug (Apple's source level symbolic debugger) and SoftwareFPU (a math co-processor emulator). Also included is Absoft's Macintosh Runtime Window Environment (MRWE) application framework (with fully documented source code as examples) and MIG graphics library. MacFortran II v3.3 features improved 68040CPU support and is fully compatible with Power Macintosh under emulation. Documentation includes special sections devoted to use of MacFortran II with the MPW editor and linker, implementation of System 7 features, and porting code to the Macintosh from various mainframes and Unix workstation platforms. \$595



NEW!

BASIC for the Newton is BASIC for the Newton! From NS BASIC Corporation, it is a fully interactive implementation of the BASIC programming language. It runs entirely on the Newton — no host is required. It includes a full set of functions and data types, hand-written input, windows, buttons and extensions to take advantage of the Newton environment. Applications can create files or access the built-in soups. Applications can also access the serial port for input and output. Work directly on the Newton, or through a connected Mac/PC and keyboard. NS BASIC includes a 150 page pocket sized manual. \$99



SmalltalkAgents™, a superset of the Smalltalk language, is fully integrated with Macintosh, incorporating design

features specifically for the RISC and Macintosh System 7 architecture. SmalltalkAgents is a true object oriented workbench that includes an incremental and extensible compiler, an array of design and cross reference tools, pre-emptive interrupt driven threads and events, an extensive class library including classes for general programming, classes for the Macintosh user interface and classes for the Macintosh operating system. Integration of components in enterprise systems is simplified with the network, telecommunication, and inter-application communication libraries. The SmalltalkAgents' extensive class library and add-on components make it especially well suited as a development workbench for custom applications in business, education, science, engineering, and academic research. \$695

SYMANTEC™

Symantec C++ for Macintosh is an object oriented development environment designed for

Want more product info? Call us at 310/575-4343.

professional Macintosh programmers. Symantec C++ features powerful object-oriented development tools within a completely integrated environment. The C++ compiler, incremental linker, THINK Class Library, integrated browser, and automatic project management give Symantec C++ fast turnaround times. This product supports multiple editors and translators, so you can use your favorite tools and resource editors as well as scripts you've written within the environment. And with ToolServer, you'll be able to customize menus and attach scripts based on Apple events, AppleScript, and MPW Tools. The built-in SourceServer provides a source code control system, allowing teams of programmers to solve tough problems faster. With SourceServer, you'll always know you're working on the latest version. And you'll have old versions at your fingertips when code "breaks" and you need to look back at modifications. Product Contents: Three high density disks, an 832-page user manual, a 568-page THINK Class Library and a 100-page C++ Compiler Guide. \$369

THINK C by Symantec Corporation. THINK C is easy to use and highly visual, making it the No. 1 selling Macintosh programming environment. Enhancements make this product faster and more versatile than ever, improving your productivity with more powerful project management, a full set of tools, and script support for major script-based languages. With the THINK environment, you spend less time on routine programming tasks due to an extremely fast compiler and incremental linker. In addition, the automatic project manager saves you time by tracking changes to your files and recompiling only those that have changes. All the tools you need — a multi-window editor, compiler, linker, debugger, browser, and resource editor — are completely integrated for speed and ease of use. One of the most valuable of these tools is the THINK Class Library, a set of program building blocks that gives you a head start in writing object-oriented applications. And with the new open architecture, you can use your favorite tools, resource editors, and scripts within the environment. THINK C is the logical next step for programmers who have worked in HyperCard or other script-based development environments. The environment supports AppleScript, Apple events, and Frontier, so you can link and automate complex, multi-project operations. Product Contents: Four Macintosh disks, an 832-page user manual, and a 568-page THINK Class Library Guide. \$219

THINK Pascal v. 4.0 by Symantec Corporation. Professionals and students will welcome this version of THINK Pascal. It is fully integrated for rapid turnaround time and lets you take advantage of System 7 capabilities. Features include support for large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking. Product Contents: Four Macintosh disks, a 562-page user manual, and a 498-page object-oriented programming manual. \$169

UTILITIES

BEdit 3.0 from Bare Bones Software is now Accelerated for Power Macintosh. This powerful, intuitive text editor offers integrated support for THINK C 7.0, THINK Reference 2.0 and MPW ToolServer. BEdit's many features include: Integrated PopUpFuncs™ technology for speedy navigation of source code files, unique "Find Differences" command (BEdit can find differences between projects and folders as well as files), support for Macintosh Drag

and Drop for editing and other common tasks, PowerTalk support for reading, sending and composition of PowerTalk mail, scripting via any OSA compatible scripting language including AppleScript and Frontier 3.0, and fast search and replace with optional "grep" matching and multi-file searching. BEdit's robust feature set and proven performance and reliability make it the editor of choice for professionals and hobbyists alike. \$99

C Programmer's Toolbox/MPW Rev. 3.0 by MMCAD. The C Programmer's Toolbox provides a wealth of programming and documentation support tools for developers who are creating new code, porting existing code, or trying to improve and expand existing code. The tools include: CDecl composes and translates C/C++ declaration statements to/from English; CFlow™ determines program function hierarchy, runtime library contents, function/file interdependencies and graphs all or part of a program's functional structure; CHilite™ highlights and prints C/C++ files; CLint™ semantically checks multiple C source files, identifying potential programming bugs; CPrint™ reformats, beautifies and documents C/C++ source files; and more... Works with MPW C/C++, THINK C, requires Apple's MPW. \$295

CLimate by Orchard Software is a command line interface that lets you communicate with your Macintosh using English commands to create, delete, rename, and move files and folders. It can start applications, format disks, restart your computer, etc. CLimate supplements the Finder. It includes a BASIC interpreter that lets you script your Macintosh without AppleScript. The interpreter includes advanced programming constructs: repeat loops, if/then/else conditionals, subroutine calls, etc... CLimate implements wildcard characters, enabling you to work on groups of files. Use CLimate instead of MPW to manage your projects. CLimate is an application occupying 70K disk space. It comes bundled with sample programs and full documentation. \$59.95

CMaster 2.0 by Jersey Scientific installs into THINK C 5 / 6 / 7 and Symantec C++ for Macintosh, and enhances the editor. Use its function popup to select a function and CMaster takes you right to it. Other features include multiple clipboards and markers, a Function Prototyper, and a GoBack Menu which can take you back to previous editing contexts. Almost all features bindable to the keyboard, along over a hundred keyboard-only features like "Add New Automatic Variable." Glossaries, AppleScript and ToolServer support, Macros, and External Tools you create too! \$129.95

Cron Manager by Orchard Software implements the UNIX Cron facility. It can open any Macintosh file on a given date and time. By creating an alias, renaming it to the date and time to open, and moving it into the special Cron Events Folder, Cron Manager will open it. Cron Manager is a control panel that creates the special Cron Events Folder inside your System Folder. It is completely transparent to the user. It works like the Startup Items folder, only smarter. It works with any Macintosh file: if you can double-click to start it, Cron Manager can open it. \$26.95. Cron Manager bundled with CLimate, \$59.95

Dialog Maker by Electric Software Corporation. Migrating from C to C++? Dialog Maker can ease your transition. Dialog Maker is an object-oriented programming library for MPW C++ and Symantec C++ (MPW and Symantec Development Environment versions) which contains a complete set of routines that create a high level interface to dialogs. Dialog Maker provides a

MAIL ORDER STORE

small number of simple, yet powerful routines to access and manipulate dialogs. Resources are used to control the most common dialog behavior allowing you to develop your application lightning fast. Minimum requirements System 7.0, MPW 3.2, MPW C++ 3.2, or Symantec C++ 6.0. \$149

InstallerPack™ by StepUp Software is a package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts. Compression formats supported are Compact Pro & Diamond. Each atom also available separately. \$219

Last Resort Programmer's Edition records every keystroke, command key and mouse event (in local coordinates) to a file on your hard disk. This is especially useful for program testing & debugging, and for technical support and help desks. If something goes wrong (because of a power failure, system crash, forgetting to save or deleting lines) and you lose a word, phrase, or document you can look in the Last Resort keystroke file and recover what you typed. Last Resort is also useful for technical support personnel, when they have to ask "What was the last thing you did before...?" \$74.95

LS Object Pascal CD includes the world's first Object Pascal compiler for Power Macintosh. 100% compatible with Apple's MPW Pascal, LS Object Pascal combines the best of Apple's native development tools with innovative new technology developed at Language Systems. Compiler options specify 68K or native PowerPC code generation. Included on the CD are: LS Object Pascal compiler, Universal Pascal Toolbox interfaces, fully loaded MPW 3.3.1, 68K and PowerPC source debuggers, PowerPC assembler, online documentation, Macintosh Tech Notes, and a special version of AppMaker by Bowers Development that generates native Pascal source code. The beta release includes upgrades to v1.0 when it becomes available. \$399

Spellswell 7 1.0.4 is an award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and Fair Witness). Spellswell 7 checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double double word errors, abbreviation errors, mixed case errors, extra spaces between words, a/an before vowel/consonant, etc... MacTech orders include developer kit with Writswell Jr., a sample Apple Events Word Services word-processor and its source code. \$74.95

MacAnalyst by Excel Software supports software engineering methods including structured analysis, data modeling, screen prototyping, object-oriented analysis, and data dictionary. This language independent tool is used by system analysts and software designers. Demo \$79, Product \$995

MacAnalyst/Expert by Excel Software supports software engineering methods with the capabilities of MacAnalyst plus state transition diagrams, state transition tables, decision tables and process activation tables. An integrated requirement database provides traceability from requirement statements to analysis or design diagrams, code or test procedures. This tool is well suited to the analysis and design of real-time or requirements driven projects. Demo \$79, Product \$1595

MacDesigner by Excel Software supports software engineering methods including structured design, object-

Want more product info? E-mail us at productinfo@xplain.com

MAIL ORDER STORE

oriented design, data dictionary and code browsing. This tool is well suited to detailed design or maintenance of software development projects. Demo \$79, Product \$995

MacDesigner/Expert by Excel Software supports software engineering methods with the capabilities of MacDesigner plus multi-task design. An integrated requirement database provides traceability from requirement statements to design diagrams, code or test procedures. This tool is well suited to design or maintenance of real-time, multi-tasking software projects. Demo \$79, Product \$1595

MacA&D by Excel Software combines the capabilities of MacAnalyst/Expert and MacDesigner/Expert into a single application. It supports structured analysis and design, object-oriented analysis and design, real-time extensions, task design, data modeling, screen prototyping, code editing and browsing, reengineering, requirement traceability, and a global data dictionary. Demo \$149, Product \$2995



MacWireFrame by Amplified Intelligence. Create your own virtual reality application with MacWireFrame, a virtual reality application frame work. Includes a

complete library of object oriented graphics routines, its own easy to understand application frame work (similar to MacApp or TCL but a lot easier to understand), plus an example application program that lets you start solid modeling right away. Comes complete with fully documented source code. All new purchases will be guaranteed a \$49.99 upgrade to the soon to be released, scriptable, MacWireFrame 5.0. Due to the overwhelming response the special price offer has been extended for a little while longer. **Special Offer: \$200.00 \$75!!!!**

McClint™ Rev. 2.2 by MMCAD. McClint locates questionable C programming constructs, saving you hours by identifying programming mistakes and latent programming bugs. Some of the checks include variable type usage, conditional and assignment statement usage, arithmetic operations in conditional expressions, misplaced semicolons, pointer type coercion, function argument passing (with and without function prototypes), local and global variable initialization and usage, and existence/shape of return statements. McClint includes a THINK C like, multiple window editor and source code highlighting system in a fully integrated environment. One or more files can be analyzed in an interactive or batch fashion. Works with THINK C (including OOPS), MPW C,... \$149.95

McCPrint™ Rev 2.2 by MMCAD. McCPrint reformats and beautifies C and C++ source code in a user specified manner. You can transform code to and from your programming style, making source code easier to read and work with. In addition to code formatting, documentation support aids include source code pagination, line number inclusion and control flow graphing. McCPrint includes a multiple window editor and source code highlighting system in a fully integrated environment. Works with THINK C, MPW C/C++, supports System 7 and 32 bit addressing for use on any system including the Quadras. \$99.95

The Memory Mine™ by Adianta is a stand alone debugging tool for Macintosh and native PowerPC. Programmers can monitor heaps, identify problems such as memory leaks, and stress test applications. Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is

shown, as are heap corruption, fragmentation, and more... Allocate, Purge, Compact, and Zap memory let users stress test all or part of a program. Source code is not needed to view heaps. It works on Macintoshes with 68020 or later and System 7.0 or later. \$99

p1 Modula-2 V5.1 is a full implementation of the ISO Standard for Modula-2 which includes exception handling, termination, complex numbers, value constructors, a standard library and more. In addition it supports objects and MacApp, foreign language calls, all current MPW interfaces, optimized 680x0 instructions, three floating point types with four modes of operation, etc. A symbolic window debugger, several utilities and a set of examples (including MacApp tutorial) are included. p1 Modula-2 requires MPW. It is targeted for professional development and prompt technical support by e-mail or FAX is granted. \$395, corporate 5 pack \$1175



PictureCDEF by Paradigm Software is a professional-level CDEF for creating custom "puffy" graphical buttons (8-64 pixels in size). PictureCDEF is multi-monitor and bit-depth sensitive. The button graphic (created with ResEdit) can be changed at runtime and even animated with a call-back routine. Create distinct buttons in six variations: PushButton, FlexiButton, ToggleButton, ChkButton, PushPicButton and TogglePicButton. Position the optional button title at left, bottom or right, or follow the system text direction for international support. 25 button frames, manual and sample code included. MacApp 3.0 support. Demo available on request. Full source code: \$95. Object code only: \$45.

Qd3d/3dPane/SmartPane source code bundle by Vivistar Consulting. **Qd3d 2.0:** Full featured 3d graphics. Points; lines; polygons; polyhedra; Gouraud shading; z-buffering; culling; depth cueing; parallel, perspective, and stereoscopic projections; performance enhancing "OnlyQD" and "Wireframe" modes; full clipping; pipeline access; animation and model interaction support; and a "triad mouse" to map 2d mouse movement to 3d. **3dPane 2.0:** Integrates Qd3d with the TCL and provides a view orientation controller. **SmartPane:** Offscreen image buffering, flicker free animation, and QuickTime movie recording. For use with Qd3d/3dPane or in 2d settings. All work with C++ compilers or ThinkC 6. \$192

NEW! QC™ by Onyx Technology, is a system extension that stress tests code during runtime for common and not-so-common errors. Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking. QC is extremely user friendly for the non-technical tester yet offers an API for programmers who want precise control over testing. \$99

QUED/M 2.6 by Nisus is the text editor that has become the industry standard for speed and efficiency. This 32-bit clean version fully supports the MPW ToolServer through Apple Events, and the CODE module feature allows you to implement external source code as a menu command. In addition, QUED/M's macro facility lets you create customized menu commands. We've even improved our acclaimed Find and Replace feature which can search unopened files for literal text or regular expressions created with GREP metacharacters. New features include a file comparison option which combines two or three files into one and marks conflicts automatically. \$149

ScriptGen Pro™ by StepUp Software is an Installer

script generator which requires no programming or knowledge of Rez. Supports StepUp's InstallerPack, StuffIt compression, custom packages, splash screens, network installs, Rez code output, importing resources, and AppleEvent link w/MPW: \$169

SoftPolish by Language Systems is a development tool that helps software developers avoid embarrassing spelling errors, detect incorrect or incompatible resources and improve the appearance of their Macintosh software. SoftPolish examines application resources and reports potential problems to a scrolling log. Independent of any programming language or environment, SoftPolish improves the quality of any Macintosh program. \$169


NEW! Spyer by InCider is a simple operated tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed. The recorded data can be saved in files for future use. Spyer works as a background process with any Macintosh application and is triggered by user defined Hot Keys. Spyer enables the "Continuous Redo" utility and is especially useful for software testing and demonstration. \$39

StoneTable by Stone Tablet Publishing: a library replacing all functions found in list manager plus: variable size columns/rows; different font, size, style, foreground color, background color per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for THINK C, THINK Pascal, MPW C, MPW Pascal. (all prices per developer) \$150, any 2 compilers \$200, any 3 compilers \$250, all 4 compilers \$300

Stone Table Extra: additional functions for StoneTable. Drag selected cells within table or to other tables; optionally add rows as part of drag; popup menus in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells. Requires StoneTable. (all prices per developer) \$50, any 2 compilers \$75, any 3 compilers \$100, all 4 compilers \$150

ViperBase by Viper Development is a fast database designed for developers that want speed but don't want to spend months or years developing a commercial quality database. ViperBase: Unlimited Records, Variable Length: \$59. ViperBase II: ViperBase + Multiple Indices. \$119





DON'T FORGET!

To receive information on any products advertised in this issue, send your request via Internet: productinfo@xplain.com

Want more product info? Call us at 310/575-4343.

LIST OF ADVERTISERS

Absoft	51
Accusoft	15
ACI US, Inc.	33
Adianta Inc.	25
Aladdin Knowledge Systems Ltd.	13
Apple Developer University	34
Ariel Publishing	87
BareBones Software	50
Celestin Company	34
Creative Solutions	25
DataPak Software, Inc.	35
dtF Americas, Inc.	36
Emergent Behavior	67
Graphic Magic	64
Graphical Business Interfaces Inc.	26 & 27
InCider International	59
Jasik Designs	17
Jersey Scientific	35
Language Systems	34
Lextek International	67
MacTech CD-ROM, Vol. 8	88
MacTech Mail Order Store	52
MacWorld Exposition	41
MacXperts	87
Manzanita Software	44
Mathemaesthetics	1
Metrowerks	8
MindVision Software	20
Neologic Systems	24
Nisus	66
Onyx Technology	64
Options Computer Consulting	59
PACE Anti-Piracy	49
POET Software	5
Quasar Knowledge Systems	IBC
Rainbow Technologies	6 & 7
Ray Sauers Associates	53
Richey Software Training	65
Scientific Placement	87
Sierra Software Innovations	BC
SNA, Inc.	39
Supersoft	54
Symantec	IFC
Trio Systems USA	45
TSE International	29
Vermont Database Corporation	31
Water's Edge Software	62
Working Software	65

LIST OF PRODUCTS

Accusoft Image Format Library • Accusoft	15
Apprentice • Celestin Company	34
BEdit 3.0 • BareBones Software	50
4D TOOLKIT 2.0 • Options Computer Consulting	59
Fortran 77 Development Tool • Absoft	51
Cataloger™ • Graphical Business Interfaces Inc.	26 & 27
C-Index/II • Trio Systems USA	45
CMaster® • Jersey Scientific	35
CodeWarrior™ • Metrowerks	8
CXBase Pro • TSE International	29
Database Scripting Kit™ • Graphical Business Interfaces Inc.	26 & 27
The Debugger V2 • Jasik Designs	17
Developer Vise 3.0 • MindVision Software	20
DragInstall 1.5.3 • Ray Sauers Associates	53
Relational Database System • dtF Americas, Inc.	36
FlexWare® • Manzanita Software	44
IcePick 3.0™ • Sierra Software Innovations	BC
Inside Out II® • Sierra Software Innovations	BC
LS Object Pascal/PPC • Language Systems	34
MacForth Plus 4.2 • Creative Solutions	25
MacHASP® • Aladdin Knowledge Systems Ltd.	13
MacEncrypt™ • PACE Anti-Piracy	49
MacNosy • Jasik Designs	17
MacRegistry™ • Scientific Placement	87
MacWorld Expo • MacWorld Exposition	41
neoAccess™ • Neologic Systems	24
Object Master • ACI US, Inc.	33
OEM Spellswell • Working Software	65
OP2CPLUS • Graphic Magic	64
p2cii™ • Sierra Software Innovations	BC
PAIGE™ • DataPak Software, Inc.	35
PatchWorks™ • SNA, Inc.	39
Pinnacle Relational Engine • Vermont Database Corporation	31
POET Object Database for C++ • POET Software	5
Power PC Training • Apple Developer University	34
Programmer Training • Richey Software Training	65
QC: The Macintosh Testing Solution • Onyx Technology	64
QDFX™ • Ariel Publishing	87
QUED/M 2.7 • Nisus	66
QuickApp™ • Emergent Behavior™	67
Recruitment • MacXperts	87
Recruitment • Scientific Placement	87
Resorcerer® • Mathemaesthetics	1
SALESBASE 2.1™ • Sierra Software Innovations	BC
Sentinel™ • Rainbow Technologies	6 & 7
Sierra Consulting Group • Sierra Software Innovations	BC
SmalltalkAgents® • Quasar Knowledge Systems	IBC
Spellchecker Toolkit • Lextek International	67
Spellswell 7 • Working Software	65
Spyer • InCider International	59
SuperPlot Pro™ • Supersoft	54
Symantec C++ 7.0 • Symantec	IFC
Template Constructor™ • Graphical Business Interfaces Inc.	26 & 27
The Memory Mine™ • Adianta Inc.	25
Tool Plus™ 2.5 • Water's Edge Software	62



TIP OF THE MONTH

TRAPPING NON-TRAP VECTOR CALLS

The Debugger has certain limitations in its ability to trace A-Line traps and such. As an example, suppose we're interested in breaking on calls to CacheFlush. It's a trap vector which is usually referenced by a JSR ([xx]) instead of the usual A-Line Trap vector. Because it doesn't go through the normal A-line dispatcher, we can't break on references to it with the Trap Intercept mechanism.

Here's some example code that creates a dummy procedure in an application, and an initialization proc that patches the low-memory trap vector so that it points to the dummy procedure. I modify the dummy procedure to make it a JMP to the original value of the low memory vector [Remember, this is for debugging, not for shipping code. Shipping code should generally **not** modify code it's about to execute – Ed stb]. Finally, the doPatch proc undoes the patch and restores the system to its previous state.

To use this, set a breakpoint at my_doPatch_Proc. When you drop into your debugger, look at the stack to see who's calling. After some looking, you can automate the process by observing where the interesting return addresses are on the stack with an action clause (such as the one shown here) that would list them in the -Notes- window:

```
?ra := (ra7)^; {return addr is contents of A7}
{ In the next line we check PC for an address in the Quadra 900 ROM }
if ?pc = 40887824 then ?ra := (ra7+46+28)^-2; {# = decimal}
writeln(?ra:ProcPtr); {display the address as a proc name + offset}
resume;

procedure my_doPatch_Proc; {make it at least 6 bytes long}
begin
end;

procedure doPatch(doit:Boolean);
CONST jCacheFlush = $6f4;
type jmpL = record opc:integer; addr:Longint; end;
VAR
  q:^jmpL;
  p:^Longint;
begin
  p := pointer(jCacheFlush); {$06F4 is the low-mem global jCacheFlush}
  q := @my_doPatch_Proc;
  if doit then begin
    with q^ do begin
      opc := $4EF9; {JMP.L}
      addr := p^;
    end;
    p^ := Ord(q);
  end
  else begin
    p^ := q^.addr; {undo patch}
  end;
end;

end;
```

– Steve Jasik, Menlo Park, CA

Send us your tips! We'll send you money, and developers all over the world will marvel at your insight, your wisdom, or the simple fact that you've got enough extra time to write and send a little bit of e-mail to make their lives a little bit better.

We pay \$25 for every tip we use, and \$50 for Tip of the Month. You can take your award in orders or subscriptions if you prefer. Make sure code compiles, and send tips by e-mail; editorial@xplain.com is a great address. See page two for our other addresses.

EVEN BETTER DATA GATHERING

The September issue's tip-of-the-month recommended a way to log standard information during crashes using the EveryTime MacsBug macro. This technique is particularly useful for beta sites to report bugs, but there's an even easier way. Many versions of MacsBug already have a built-in macro called StdLog that does almost the same thing. Beta-sites can just use stdlog to supply you with information and it doesn't require them to muck with their "Debugger Prefs" file.

– Harold Ekstrom, Walnut Creek, CA

The Future is Dynamic, Is Your Development Environment?

- ◆ Dynamic environment with incremental compiling. NO compile-link downtime.
- ◆ Intelligent, integrated UI Components. NO dumb resources separate from your code.
- ◆ Integrated Object-Linker provides component support today, NOT tomorrow.
- ◆ High-level support for System 7.x makes it easy to leverage Apple technology.
- ◆ Simplified access to databases via DAL/DAM. Sample code included.
- ◆ Inline foreign function callbacks to C, Pascal, Fortran and Assembly.
- ◆ Fully styled source code with fonts and color. WorldScript™ support included.
- ◆ HyperCard™ XMCD interface to assist HyperCard developers "moving up."
- ◆ Cross-platform development with full support of native OS functionality.
- ◆ Scalable development, from simple utilities to complex distributed applications.

SmalltalkAgents® is *the* Interactive OODL

SmalltalkAgents® is the *premier* commercial object-oriented dynamic development environment. New key features of SmalltalkAgents Professional include:

NEW Documentation

Features the Macintosh® version of Dan Shafer's *Smalltalk Programming for Windows* book, specifically re-written for SmalltalkAgents, and four other complete volumes including "Road Map", "A Quick Trip to ObjectLand", "Developer Guide: Workbench", and "Developer Guide: Class Frameworks".

NEW GUI Builder

Live "Drag and Drop" manipulation to build your application's visual interface using smart components that "know" how to behave. Wire together components and save designs as reusable templates that can be dropped into other applications or components.

NEW External Code Linking Toolkit

Preserve your investment in existing C, Pascal, and Fortran code by dynamically linking your routines into the SmalltalkAgents integrated development environment, including optional object "wrappers".

**SmalltalkAgents Lite
now available for \$95
direct from QKS!**



**Buy SmalltalkAgents Pro now for discount
on native Power Macintosh version!***
(*Shipping January 1995, offer ends 12/31/94)

*The NEXT GENERATION Tool for Cross-Platform Object-Oriented
Programming, including a superset of Smalltalk...
the language of choice for corporate development!*

1-800-296-1339
or 301-530-4853 (info@qks.com)



Sierra Software Innovations

Six Years of Quality Products and Services

SALESBASE 2.1™

Fully Integrated Sales Automation Solution for Large Corporations

Inside Out II®

Industrial Strength Multi-User Relational Database Engine

p2cII™

Advanced Object Pascal to C++ Source Code Translator

IcePick 3.0™

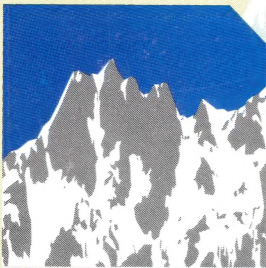
Original Award Winning MacApp 2.x and 3.x View Editor

Sierra Consulting Group

Top-Notch DataBase and SalesAutomation Consulting

Please call 800.621.0631 for a complete information package
on Sierra's line of prestigious products and custom services.

Better Software through Innovation



**SIERRA
SOFTWARE
INNOVATIONS**

923 Tahoe Blvd., Incline Village, Nevada 89451

Phone: 702.832.0300 Fax: 702.832.7753 Internet: D2086@applelink.apple.com

SALESBASE, Inside Out II, p2cII, IcePick are trademarks of Sierra Software Innovations.